Future Generation Computer Systems 29 (2013) 728–738

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Monte Carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies

Sorina Camarasu-Pop *, Tristan Glatard, Rafael Ferreira da Silva, Pierre Gueth, David Sarrut, Hugues Benoit-Cattin

Université de Lyon, CREATIS; CNRS UMR5220; Inserm U1044; INSA-Lyon; Université Lyon 1, France

ARTICLE INFO

Article history:
Received 10 February 2012
Received in revised form
31 August 2012
Accepted 3 September 2012
Available online 8 September 2012

Keywords:
Grid computing
Monte Carlo
Merge
Checkpointing
Workflow
Dynamic parallelization

ABSTRACT

This paper introduces an end-to-end framework for efficient computing and merging of Monte Carlo simulations on heterogeneous distributed systems. Simulations are parallelized using a dynamic load-balancing approach and multiple parallel mergers. Checkpointing is used to improve reliability and to enable incremental results merging from partial results. A model is proposed to analyze the behavior of the proposed framework and help tune its parameters. Experimental results obtained on a production grid infrastructure show that the model fits the real makespan with a relative error of maximum 10%, that using multiple parallel mergers reduces the makespan by 40% on average, that checkpointing enables the completion of very long simulations and that it can be used without penalizing the makespan.

1. Introduction

Monte Carlo simulations are employed in several scientific domains due to their capacity to produce realistic results from repeated sampling of events generated by pseudo-random algorithms. Accurate results require a large number of statistically independent events, which has a strong impact on the computing time of the simulation. Nevertheless, Monte Carlo simulations are easily parallelized using a split and merge pattern. They can thus exploit important amounts of resources available in world-wide computing grids, such as the European Grid Infrastructure (EGI)[1] and Open-Science Grid.[2]

Various strategies have been proposed to efficiently execute Monte Carlo simulations on distributed platforms [1–3], but they mostly focus on the computing part of the simulation while the merging of partial simulation results is also a critical step. Merging partial results is a data-intensive operation which is very sensitive to the number and production date of partial results, to the availability of storage resources, and to the network throughput and latency. On world-wide computing infrastructures, partial results are usually geographically distributed close to their production site, which exacerbates the cost of data transfers. In some cases, the merging time can even become comparable to the simulation makespan.

Software heterogeneity observed in large computing infrastructures is also a show-stopper due to the amount of errors it generates. Job failure rates of more than 10% are commonly observed in EGI [4], with strong consequences on the performance of applications. Checkpointing is often used to improve application reliability, but it has to be properly tuned to limit overheads. Checkpointing is also worth studying to address result merging because it enables incremental production of partial results during the computing phase.

Performance analysis and modeling are important to understand the behavior of distributed applications and to help tune parameters such as the checkpointing delay. But they remain a challenge for applications running in production conditions because most parameter values are unknown before the completion of the experiment, for instance the background load of the infrastructure or the characteristics of the resources involved in the execution. Models used in production have to be able to cope with such a lack of information, focusing on parameters that are measurable by the applications.

In this paper, we propose an end-to-end Monte Carlo simulation framework combining dynamic simulation load-balancing with parallel, incremental merge of checkpointed results. The framework is designed for heterogeneous, non-reliable distributed systems deployed in production. It does not make any assumption on the nature or characteristics of computing resources. We assume

* Corresponding author.
E-mail address: sorina.pop@creatis.insa-lyon.fr (S. Camarasu-Pop).
1 http://www.egi.eu
2 http://www.opensciencegrid.org

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.09.003

that it is deployed in the application space, i.e., we focus on improved usage of existing infrastructures rather than on the design or tuning of their services. In particular, job scheduling and data placement are assumed to be controlled by infrastructure services. We also propose a model to analyze the behavior of the proposed framework and help tune its parameters.

Related work on parallel Monte Carlo simulation, on merging, and on checkpointing techniques is presented in Section 2. Then we describe the proposed framework integrating parallel, incremental merge with simulation checkpointing and dynamic load-balancing in Section 3. In Section 4, experiments evaluate the impact of merging strategies and checkpointing in production conditions. The paper closes on a discussion and conclusions.

2. Related work

This section presents related work on Monte Carlo parallelization methods and their limitations concerning partial result merging when used on distributed environments. It also explains the MapReduce paradigm which is a common way of implementing split-and-merge patterns. An outline on checkpointing is finally presented.

2.1. Parallel Monte Carlo

The parallelization of Monte Carlo algorithms is widely studied [5,6,1,2,7], but their proper load-balancing remains a challenge on heterogeneous unreliable platforms, where assigning the same number of events to each computing job is clearly sub-optimal. To deal with this issue, Rosenthal [5] proposes a simple but effective technique in which a master assigns a fixed computing time T to each job involved in the simulation. Jobs simulate until time T, merge their partial results and report them to the master. The master then merges the results produced by all the computing jobs. The merging phase consists here in computing a weighted average based on the number of events simulated by each job. In all cases, the total computing time and the number of available parallel nodes have to be known in advance so that T can be computed accordingly.

On grid architectures, the above technique can suffer significantly from the variable queuing times of the jobs. Moreover, if the total computing time is not easily known, then T cannot be estimated properly. New strategies are needed in these cases. Mascagni and Li [1] uses the N out of M strategy, i.e. it increases the number of submitted jobs from N to M. As soon as N results are ready, the final result can be produced. This is an example of task replication classically employed on distributed systems [8].

Camarasu-Pop et al. [9] proposes a dynamic load-balancing approach for distributed and heterogeneous infrastructures. To better exploit available resources, simulation jobs are kept running until the desired number of events is reached. Periodically, they send their number of simulated events to a master job that sums up the events and stops the computing jobs when the total number is reached. Thus each computing resource contributes to the whole simulation until the end. Only light communications are performed between the computing jobs and the master.

Such dynamic load-balancing approaches are possible since Monte Carlo algorithms are moldable, i.e. the workload Γ is freely divisible. In this case, the scheduler can dispatch arbitrarily small quantities of work to n parallel computing jobs, so that the n parallel jobs finish quasi-simultaneously. This behavior has been modeled in [10], where the simulation makespan is expressed as the average waiting time E_L plus the average running time of the n parallel jobs: $M = \frac{\Gamma}{n} + E_L$.

Most of the existing literature on parallel Monte Carlo strategies concentrates on the computation itself and says little about the merging of partial results. This problem is indeed of limited importance on local clusters. Nevertheless, when results are geographically distributed over the sites of a world-wide system, transferring and merging them may be as long as the parallelized Monte Carlo computation. The merging phase can thus represent a key element in improving the performance of parallel Monte Carlo algorithms.

2.2. MapReduce

MapReduce has become a common way of implementing split-and-merge patterns. MapReduce is a programming model and an associated implementation for processing large datasets on distributed platforms like clusters or grids [11–13]. In the Map step, the master node partitions the inputs and distributes them to Mappers for processing. During the Reduce step, the partial results are combined into the final output. Several Reducers may co-exist, each of them producing one independent result.

One of the first and most well-known implementations is the Google MapReduce library [11]. The input data is partitioned into M splits, so that each Mapper takes an input key/value pair and produces a set of intermediate key/value pairs. These intermediate results are partitioned into R pieces that are distributed to Reducers. Thus, each Reducer takes as inputs an intermediate key and a set of values for that key, and produces one output corresponding to a collection of values. After successful completion, the final output of the MapReduce execution is available in the R output files. A combiner can be used to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

In [14], where MapReduce is used for High Energy Physics and KMeans clustering, another combine strategy is used to provide a single final result. Ekanayake and Pallickara [14] use Hadoop,[3] Apache's MapReduce implementation, which relies on its own distributed filesystem to schedule the MapReduce computation tasks depending on the data locality and hence to improve the overall I/O bandwidth. Ekanayake and Pallickara [14] evaluate the total execution time and speed-up with respect to data size, but no information is given on the performance of the reduce and combine steps individually.

Map and reduce progresses are monitored separately in [15], showing that the reduce step can take much more time than the map step. In this paper the authors propose a modified MapReduce architecture allowing data to be pipelined between operators. To support pipelining, the authors modified the map task to push data to Reducers as it is produced. Because the number of Mappers far exceeds the number of Reducers, pushing data too early to Reducers may create a bottleneck. Therefore, data is buffered by the Mappers and sent to Reducers only when it reaches a certain threshold. Condie et al. [15] also propose a modified version of the Hadoop MapReduce framework that supports online aggregation. In this scheme, the system gives the user an estimate of the final query result at all times during processing. This estimate is obtained from partial results checkpointed periodically.

2.3. Checkpointing

Checkpointing consists in saving the state of a running program so that it can be reconstructed later in time [16]. It has various purposes, among which providing fault-tolerance and accessing partial results. These are valuable for parallel Monte Carlo

3 http://hadoop.apache.org/mapreduce

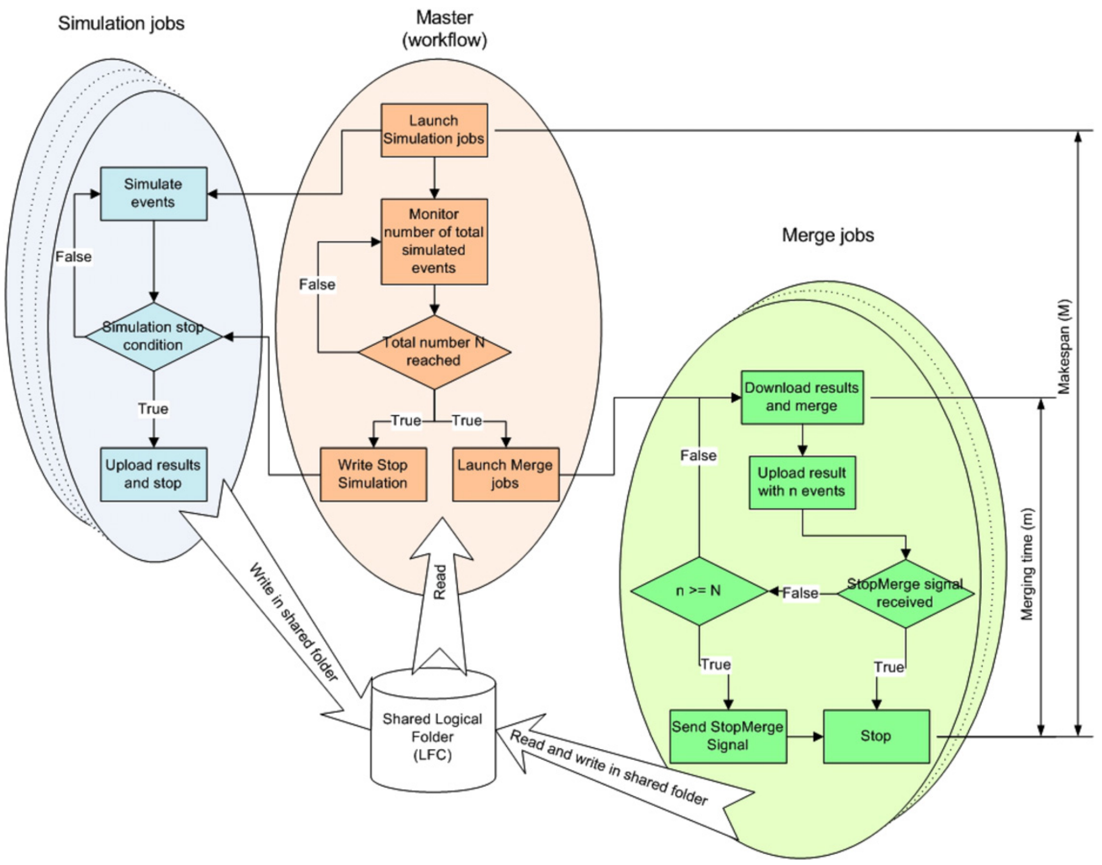


Fig. 1. Workflow without checkpointing. Simulation jobs are represented in blue, merging jobs in green and the master in orange. Multiple simulation and merge jobs run in parallel but, for readability reasons, only one simulation and one merge jobs are represented. Simulation jobs upload their results once at the end of their lifetime. The SimulationStop condition given by the master triggers: (i) results upload from simulation jobs and (ii) the launch of merging jobs. The StopMerge signal is sent by the first merging job having merged the required number of events. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

applications running on heterogeneous platforms that are prone to failures.

Some grid/cluster computing systems like Condor [3] provide integrated checkpointing mechanisms. Condor can take a snapshot of the current state of the process, including code, stack and data segments, all CPU registers, open files and signal handlers. Such process-level checkpointing mechanisms are transparent to end users, but they are platform-dependent and may not be suited for large applications requiring significant disk and network resources.

Application-level checkpointing [1] is often more efficient since the application can choose to save the minimal information required to resume computation. As described in [1], Monte Carlo applications generally need to save only a relatively small amount of information. For instance, the GATE [17] Monte Carlo simulator has a mechanism allowing the application to periodically pause and checkpoint on disk the state of its actors. The user can specify if saved results should be complete (all events simulated since the beginning of the execution) or incremental (only the difference between two successive checkpoints). After each checkpoint, the simulator automatically resumes its execution. In case a failure occurs, results can be retrieved from the last checkpoint and the simulator is re-started with a new independent random seed.

The next section describes the complete framework proposed for robust execution of Monte Carlo simulations on distributed and heterogeneous systems. A model of the makespan of the complete simulation is also introduced.

3. Proposed framework

3.1. Framework description

Our framework extends the work proposed in [9] by integrating the merging phase with the dynamic load-balancing used in the parallelization of Monte Carlo simulations. To improve performance, the merging phase is also parallelized in multiple jobs providing a single final result. Our framework also includes checkpointing to improve the robustness of long simulations.

Algorithm 1 Master algorithm for dynamic load-balancing (extracted from [9])

```
N=total number of events to simulate
n=0
while n< N do
    n = number of events simulated by running and successfully completed jobs
end while
Send stop signal to all jobs
Cancel scheduled jobs
```

The workflow without checkpointing is described in Fig. 1. The master generates multiple parallel jobs containing the total number N of events to simulate. Each job periodically transmits its current number of simulated events to the master. At this point, these events reside on the local disk and will not be available for

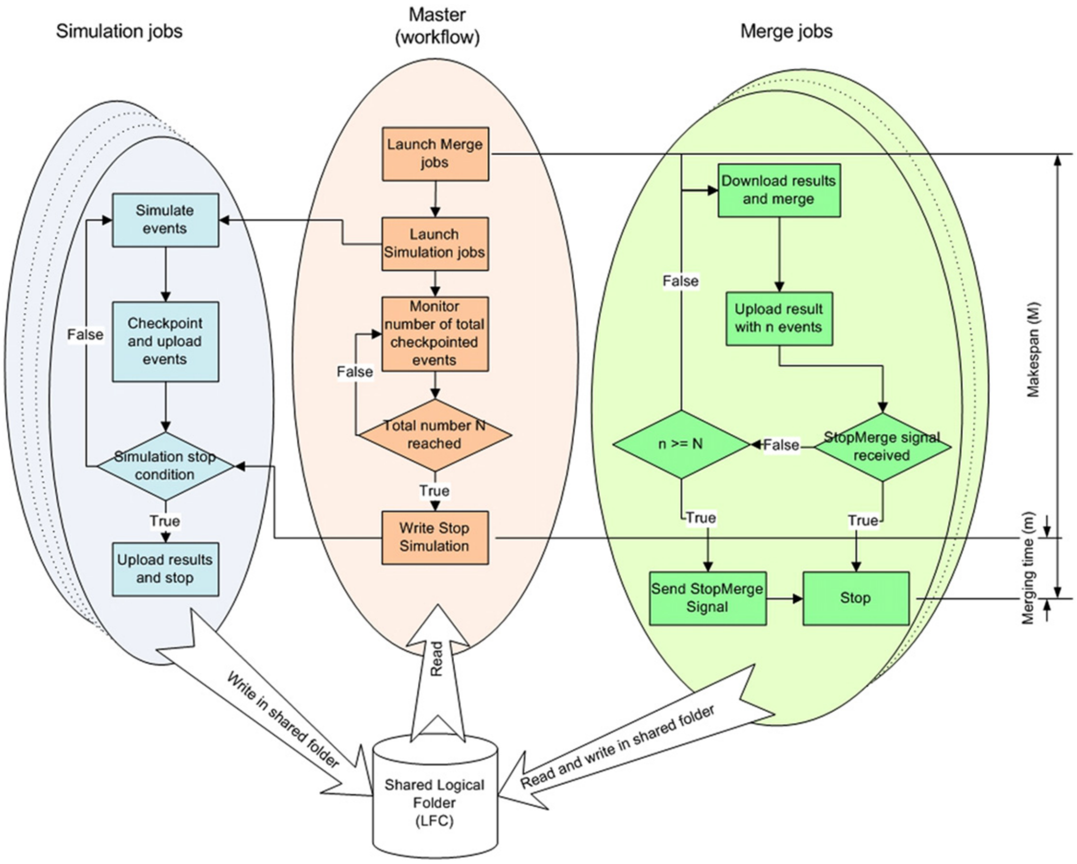


Fig. 2. Workflow with checkpointing. Simulation jobs are represented in blue, merging jobs in green and the master in orange. Multiple simulation and merge jobs are launched in parallel at the beginning of the workflow. For readability reasons, only one simulation and one merge jobs are represented. Simulation jobs upload their results regularly, at the same frequency as they checkpoint their partial results. The SimulationStop condition given by the master triggers the end of simulation jobs. From this moment on, only merging jobs continue to run during the "extra merging time". The StopMerge signal is sent by the first merging job having merged the required number of events. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Algorithm 2 Simulation jobs algorithm for dynamic load-balancing (extracted from [9])

```
N=total number of events to simulate
n=0, lastUpdate=0, updateDelay=5min
while stop signal not received AND n< N do
  Simulate next event
  n++
  if (getTime() - lastUpdate) >updateDelay then
    Send n to master
    lastUpdate = getTime()
  end if
end while
Upload results to output storage
```

merging in case the simulation job fails. Simulation jobs then check the stopping condition given by the master when the total number of simulated events from all simulation jobs is reached. This part of the workflow corresponds to the dynamic load balancing algorithm presented in Algorithms 1 and 2. When the stopping condition is reached, the master launches multiple merging jobs and the simulation jobs upload their partial results into a shared logical folder. Note that the content of this folder may be physically stored on multiple, distributed storage elements. While there is no StopMerge signal, the merging jobs select, download and merge batches of partial results. Their result is then uploaded back to the shared folder for subsequent merging. If a merge job produces a result containing the total number of events, then it also sends the StopMerge signal.

The workflow with checkpointing is described in Fig. 2. Note the following important differences with the version without checkpointing:

- Simulation jobs checkpoint and upload their partial results at the checkpoint frequency.
- The master monitors the number of checkpointed events instead of simulated events. Checkpointed events are the ones which were uploaded to the shared logical folder and are therefore available for merging.
- Merging jobs are launched from the beginning of the workflow because partial results are available since the first checkpoint. Apart from that, merging jobs are the same as without checkpointing.

In both workflows, each parallel merging job contributes to the final result by merging a fraction of the partial results. The merging process consists in (i) selecting any maximum nf files to merge, each file i containing n_i events, (ii) removing the selected files from the shared logical folder, (iii) downloading and merging the selected files, and (iv) uploading the result containing $n = \sum n_i$ events into the shared logical folder. nf is a parameter of the workflow and it is the same for all merging jobs. For a good load-balancing of the merging activity, it should be smaller than the total number of files to merge divided by the total number of mergers. The merging process is commutative and associative as an addition. As a consequence, the merged output is of the same type as the merge inputs and it is merged as any other partial result. Thus, the outputs of the merging jobs and the partial results produced by computing jobs can be merged in any order. The only constraint is that the same file must not be merged multiple times. A lock mechanism ensures that this condition is respected.

In the approach presented in this paper, the parallel computing jobs correspond to the Mappers of the classical MapReduce approach. While these Mappers receive an input subset, here they receive the complete input data (like the neuro-imaging application in [18]) plus a unique random seed which allows us to produce statistically independent partial results. Since partial results are not identified with a key as in the standard MapReduce approach, they can be all merged together in any order. Mergers can be thus independent and pick any element from the pool of available partial results. Similarly to MapReduce, several mergers can co-exist, but in our case a single final result is needed. Although here there is no central entity to manage the distribution of intermediate results to Reducers, the coordination of the mergers is ensured by the lock mechanism. Together with the StopMerge condition, it also ensures the production of one single final result.

3.2. Model

A model of the makespan of Monte Carlo executions with and without checkpointing is presented here. It mainly aims at explaining measures made in production.

3.2.1. Model without checkpointing

Monte Carlo simulations may be arbitrarily partitioned and executed as a set of independent jobs on computing resources. Since the workload Γ is freely divisible, the dynamic load-balancing algorithm behaves as if 1-event jobs were continuously distributed to n parallel simulation jobs without overhead. In this case, the simulation jobs finish quasi-simultaneously, as it can be seen on the job flow in Fig. 4. In these conditions, if runtime errors, grid submission failures, and latency variability are ignored, the makespan M is given by the average waiting time plus the average running time of the n parallel jobs: $M = \frac{\Gamma}{n} + E_L$ as explained in [10].

If we take into account that with a failure rate ρ, only $n(1-\rho)$ jobs contribute to the simulation, the model becomes: $\Gamma = n(1-\rho)(M - E_L)$, i.e. $M = \frac{\Gamma}{n(1-\rho)} + E_L$

And if we consider the merging time m in addition to the computing time, the makespan becomes:

$$M = \frac{\Gamma}{n(1-\rho)} + E_L + m \tag{1}$$

3.2.2. Model with checkpointing

If checkpointing is activated, then failed jobs may still contribute to the final result with the checkpoints realized before failing. In this case the makespan depends on F, the cumulative distribution of the time to failure (TTF) of the jobs on the target infrastructure. $F(t)$ is the probability that a job does not face any failure for a duration t. Only failures occurring during job execution are considered; system errors happening during scheduling or queuing are ignored.

Let c be the time between two consecutive checkpoints of a simulation job. We assume that c is fixed and cannot be changed during the simulation. Since simulation jobs are not synchronized due to their individual queuing time, their checkpoints are not synchronized either. The end of the simulation is given by the last checkpoint of the job which contributes enough to reach the simulation stop condition, while the other jobs are still in a computing phase.

Let k be the integer such that $kc \leq M - m - E_L < (k+1)c$. k is the number of checkpoints made by jobs which do not fail. The total CPU time Γ consumed by the simulation is the sum of the expected CPU times checkpointed by simulation jobs:

$$\Gamma = n \times \left[\sum_{i=0}^{k-1} ic\Big(F((i+1)c) - F(ic)\Big) + kc\,(1 - F(kc))\right],$$

where the first term of the sum represents the contribution of a job that checkpoints exactly i times with a probability $F((i+1)c) - F(ic)$, and the last term represents the contribution of a job that checkpoints a maximum number of k times with a probability $1 - F(kc)$. Thus:

$$\Gamma = nc \times \left(k - \sum_{i \leq k} F(ic)\right). \tag{2}$$

Note that the time between the last checkpoint ($t = kc$) and the end of the simulation does not contribute to Γ.

According to the experimental data that will be presented in Section 4.3 (see Fig. 8), it is reasonable to assume that F is linear from $t = c$ to $M - m - E_L$:

$$F(ic) = F(c) + (i-1)\frac{F(kc) - F(c)}{k-1}.$$

Thus:

$$\begin{aligned}\sum_{i \leq k} F(ic) &= kF(c) + \frac{F(kc) - F(c)}{k-1}\sum_{i=2}^{k}(i-1) \\ &= k\frac{F(kc) + F(c)}{2}.\end{aligned}$$

And from Eq. (2):

$$k = \frac{\Gamma}{nc\left(1 - \frac{F(kc)+F(c)}{2}\right)}.$$

Note that this expression is easily extended to the case where F is linear only from $t = pc$ ($p < k$). The makespan can then be expressed using the following approximation:

$$k \approx \frac{M - m - E_L}{c} - \frac{1}{2}$$

so that:

$$M = \frac{\Gamma}{n\left(1 - \frac{F(kc)+F(c)}{2}\right)} + \frac{c}{2} + m + E_L \tag{3}$$

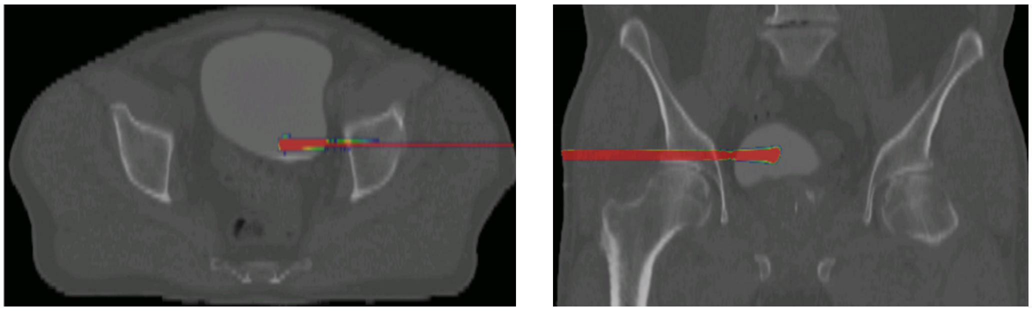
Experiments conducted both with multiple mergers and different checkpointing frequencies are presented in the next section.

4. Experiments and results

4.1. Implementation

Both scenarios, with and without checkpointing, have been implemented using the MOTEUR workflow engine [19]. Both workflows are integrated into the VIP/GATE-Lab web portal described in [20] and openly available.[4] The portal gives access to some 10 simulators and counts more than 200 registered users. Jobs are executed using DIRAC [21] pilot jobs on the resources available to the biomed virtual organization (VO) within the EGI grid.

[4] https://vip.creatis.insa-lyon.fr



(a) Patient CT, axial view. (b) Patient CT, coronal view.

Fig. 3. Example of simulation made with GATE. The figure depicts axial and coronal slices of a CT pelvis image. The dose distribution obtained by a proton pencil beam and computed by the simulation is overlayed in red. The bladder is contrast enhanced for display purposes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

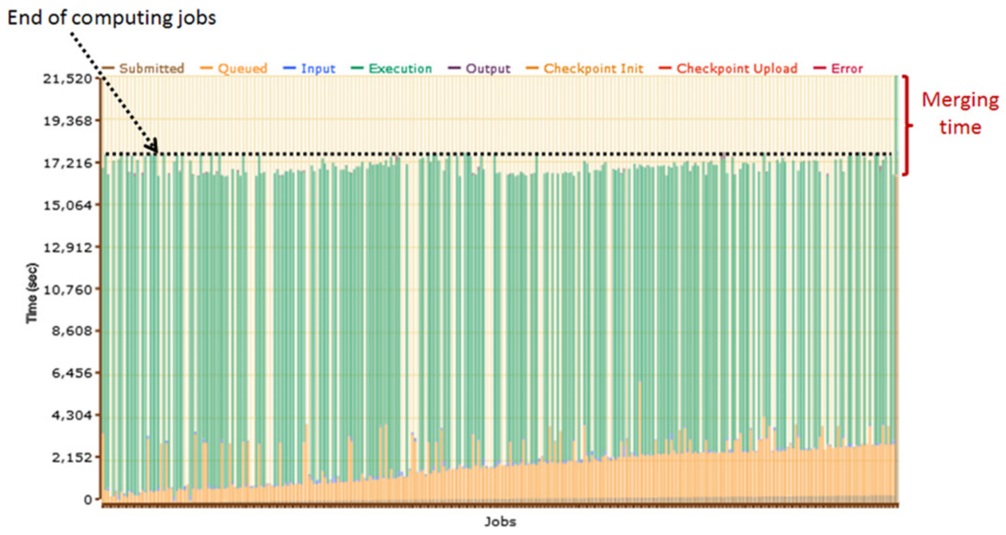


Fig. 4. Job flow for a GATE simulation without checkpointing and with one merging job. Computing jobs finish almost simultaneously, approximately at the same moment the merging job starts its execution. Empty bars correspond to jobs for which we lack information on the running and completion times (it may happen for some error, stalled or canceled jobs).

The biomed VO has access to more than 150 clusters (CE) and some 3.5 PB of storage distributed on Storage Elements (SE). Data is thus physically world-wide distributed, but a centralized logical view is provided by the Logical File Catalog (LFC). Worker Nodes (WN) executing grid jobs download input data from SEs, then compute and produce results locally. Results have to be uploaded on a SE and registered into the LFC to be available from other WNs.

Concurrent access to partial results stored in the logical shared folder is partially handled at the LFC level since listing, moving and deleting files are transactional operations. To cope with concurrency issues, the mergers lock the common results folder before selecting the files to merge, then release the lock after moving these files into their own folder. The lock is implemented with the creation of a directory in the shared logical folder, which is also a transactional operation. The lock has a limited lifetime. Therefore, if a process fails after it has acquired the lock, no deadlock is created.

4.2. Experiments

Experiments were conducted using the GATE [17] Monte Carlo simulator, which has an embedded checkpointing mechanism. GATE is a Geant4-based open-source software developed by the international OpenGATE collaboration[5] and able to perform nuclear medicine simulations, especially for TEP and SPECT imaging, as well as for radiation therapy. It is used by approximately 1200 users world-wide. The simulation used here consists in the successive stochastic tracking through matter of a large set of individual events, each event having an initial set of properties (type, location, direction, energy, etc.). For these experiments, we chose to run a proton therapy simulation. Fig. 3 shows the result produced by a 50M events simulation. It represents the dose distribution obtained by a proton pencil beam as described in [22].

Three experiments were conducted:

- The first experiment (Exp 1) aims at demonstrating the importance of using checkpointing for long simulations. For this experiment, we executed a GATE simulation of 440M events representing roughly one year of CPU time. We measured the number of simulated and merged events with and without checkpointing. Merged events are the ones that were merged at least once.
- The second experiment (Exp 2) aims at determining the impact of using multiple parallel mergers. For this experiment, we executed a GATE simulation of 50M events representing roughly 41 days of CPU. The workflow without checkpointing was used. Four different runs were performed, with 1, 5, 10 and 15 mergers.
- The third experiment (Exp 3) studies the influence of checkpointing. As for the second experiment, we executed a GATE simulation of 50M events representing roughly 41 days of CPU.

[5] http://www.opengatecollaboration.org

Table 1
Experiment results—without checkpointing (Exp 2).

Run	ρ	E_L (s)	m (s)	M real (s)	M model (s)	Model error (%)
1 merger #1	0.148	5298	19 980	41 916	38 653	7.8
1 merger #2	0.317	1404	26 760	42 460	46 166	−8.7
1 merger #3	0.180	1554	5 220	21 528	21 490	0.2
1 merger mean	–	–	17 320	35 301	35 436	–
5 mergers #1	0.187	1361	3 060	18 384	20 159	−9.7
5 mergers #2	0.191	2295	6 480	23 742	24 718	−4.1
5 mergers #3	0.140	950	9 120	25 613	23 092	9.8
5 mergers mean	–	–	6 220	22 579	22 656	–
10 mergers #1	0.102	1346	1 920	17 601	16 588	5.8
10 mergers #2	0.171	2143	2 580	21 927	19 749	9.9
10 mergers #3	0.213	3240	2 940	23 055	22 176	3.8
10 mergers mean	–	–	2 480	20 861	19 504	–
15 mergers #1	0.128	2369	4 080	21 637	21 061	2.7
15 mergers #2	0.123	2483	3 060	20 343	19 659	3.4
15 mergers #3	0.150	1580	2 460	18 326	18 406	−0.4
15 mergers mean	–	–	3 200	20 102	19 709	–

For this experiment, the workflow with checkpointing was used. Three different runs were considered, with a checkpointing frequency of 30, 60 and 120 min. In each case, 10 parallel mergers were launched from the beginning of the simulation.

For the last two experiments, we measured the total CPU time of the simulation (Γ), the mean job waiting time (E_L), the merging time (m), the failure rate of simulation jobs (ρ) and the fraction of simulation jobs that failed before the first checkpoint ($F(c)$) for the simulations with checkpointing. For the three experiments, the GATE simulation was split into 300 jobs and each merger selected maximum 10 files to merge at each merging step.

Experiments were conducted on the production platform described above. Using a production infrastructure was a deliberate choice to ensure that the realism of our assumptions could not be questioned. As a counterpart, reproducibility is limited because performance depends on the grid conditions at the time of the execution, in particular the system load. Experiments were repeated three times to capture some of the grid variability.

4.3. Results

4.3.1. Added value of checkpointing (Exp 1)

Fig. 5 shows the number of merged and simulated events along time. Merged events are events that have been processed at least once by a merger, while simulated events still reside on the local disk of a simulation job and are not available for merging in case the simulation job fails. For the workflow without checkpointing (Fig. 5(a)) the number of simulated results increases steadily during the first 24 h. Afterwards, some of the jobs are killed by sites imposing a maximal execution walltime. Indeed, approximately 30% of the job slots in the biomed VO are managed by batch queues imposing a maximal walltime of less than one day and 65% of less than 2 days. If the simulated events are not checkpointed, they are lost when the jobs are killed. Killed jobs are resubmitted, which explains why the number of events still increases even after the first 24 h. In this experiment, the workflow without checkpointing is simply not able to complete the simulation of 440M events.

Conversely, the workflow with checkpointing (Fig. 5(b)) is able to complete the simulation. Since checkpointed events are not lost when jobs are killed, their number increases steadily until the end of the simulation.

4.3.2. Impact of multiple mergers (Exp 2)

Table 1 details all measures obtained from Exp 2, and Fig. 6 compares the measured makespan with the makespan computed from the model in Eq. (1). The model correctly explains the experiments, with a relative error of less than 10%, and less than 5% for half of the simulations. Note the importance of taking into account the merging time, without which simpler models could not fit the experiments.

The results show that using a unique merger is clearly suboptimal. From Table 1 we can see that the average makespan with one merger can be reduced by 40% when using 10 parallel mergers (from 35 301 to 20 861 s). This is due to an important decrease of the merging time, which can represent more than 50% of the total makespan of the simulations with only one merger, while it represents less than 15% for the simulations with 10 parallel mergers.

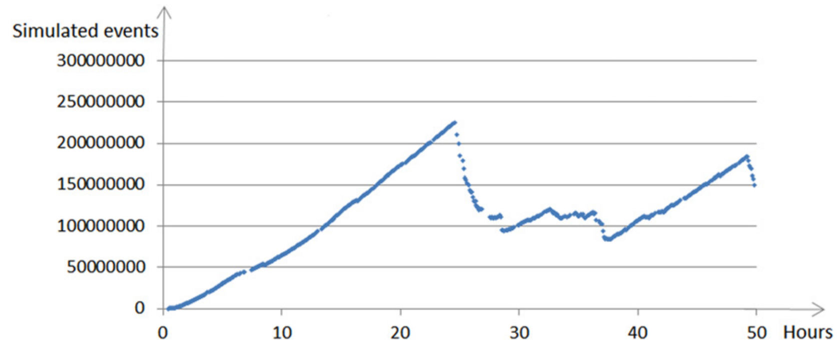
We also notice that there is a threshold above which increasing the number of parallel mergers is not useful. In our case, experiments with 10 mergers perform on average as well as those with 15 mergers.

4.3.3. Influence of the checkpointing frequency (Exp 3)

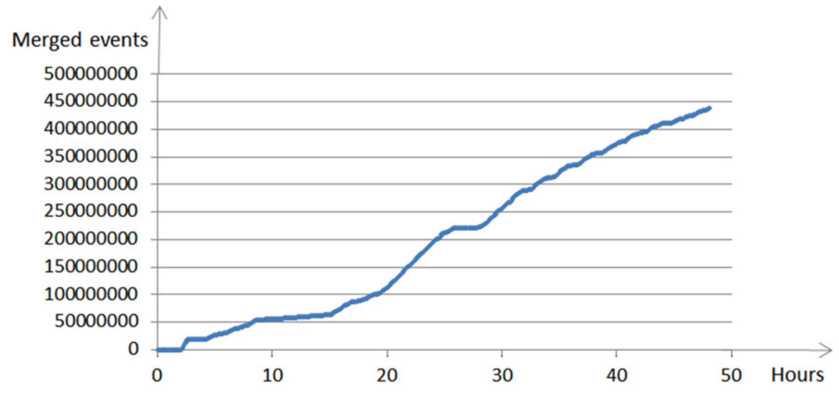
Table 2 shows measures obtained from Exp 3, and model values computed using Eq. (3), assuming $F(kc) = \rho$. Fig. 7 compares the measured and modeled makespan. Overall the model correctly explains the experiments, with a relative error lower than 10%.

From Table 2 we notice that the merging time m decreases as the checkpointing period increases. This can be explained by the fact that the parallel mergers can be saturated with partial results if the checkpoints are too frequent. At the same time, according to the model and to Eq. (3), the makespan increases with the checkpointing period. A trade-off is thus needed. Among the three runs, we notice that the checkpointing period of 60 min provides the best average makespan.

Note that the merging time with checkpointing periods of 60 min is significantly smaller than without checkpointing and 10 mergers (in average, 1260 s versus 2480 s). Indeed, as illustrated on Fig. 2, with checkpointing, the merging jobs run concurrently with the simulation jobs, which reduces the merging time. Nevertheless, despite the decrease of the merging time, the makespan of the two runs remains comparable. This is due to the checkpointing overhead introduced by the checkpointing frequency as modeled by Eq. (3) and observed on Fig. 7. This overhead could be reduced if the checkpointing frequency were not fixed and if we could force the application to checkpoint as soon as the number of simulated events is reached. In this experiment, checkpointing does neither improve nor penalize the makespan.



(a) Workflow without checkpointing.



(b) Workflow with checkpointing.

Fig. 5. Results for the experiment with a very long simulation (Exp 1), representing roughly one year of CPU time. The workflow without checkpointing (a) is not able to complete the simulation because events from killed jobs are entirely lost. For the workflow with checkpointing (b), the number of checkpointed events increases steadily till the end of the simulation.

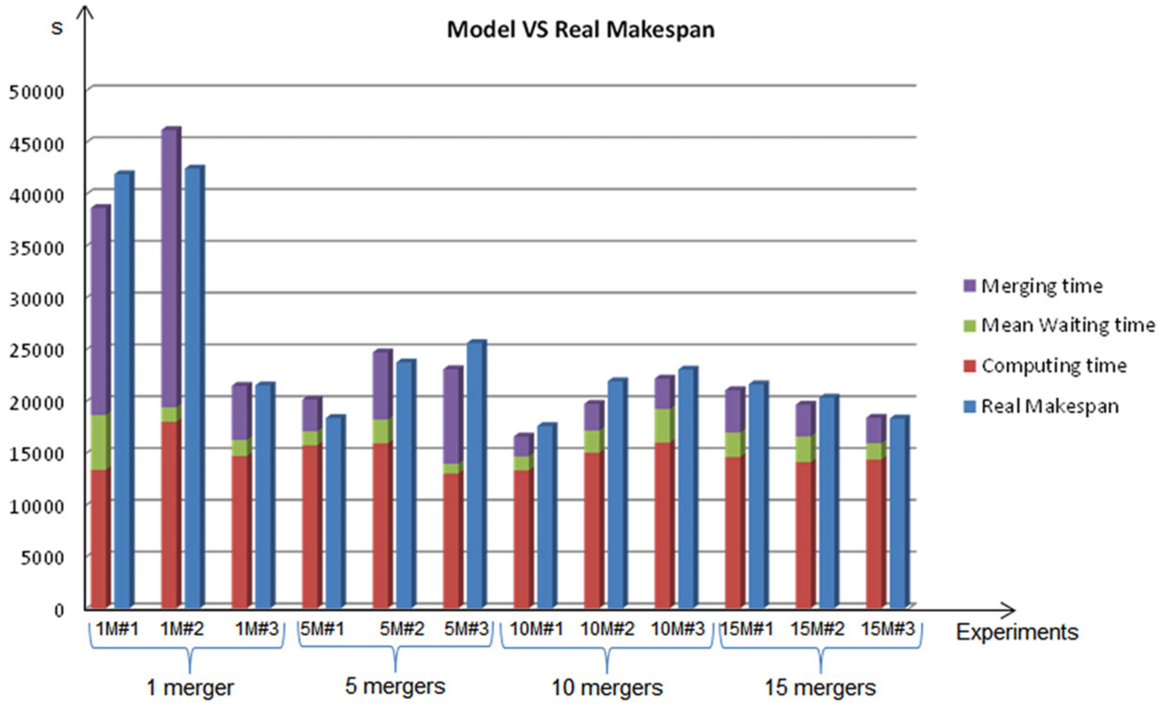


Fig. 6. Results for the experiment without checkpointing (Exp 2). For each of the twelve GATE simulations (three repetitions for each of the four runs) two bars are printed: the blue bar on the right represents the real (measured) makespan in seconds, while the stacked bar on the left represents the makespan computed using the proposed model for GATE without checkpointing. The three stacked elements correspond to the three terms in Eq. (1). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

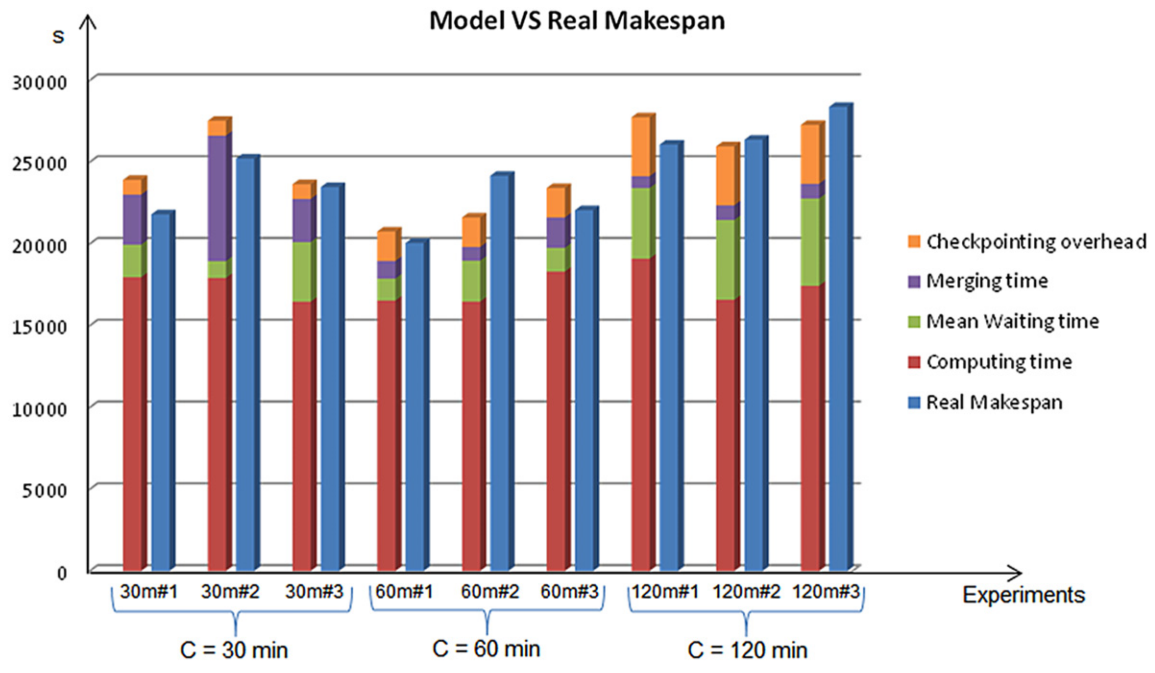


Fig. 7. Results for the experiment with checkpointing (Exp 3). For each of the nine GATE simulations (three repetitions for each of the three runs) two bars are printed: the blue bar on the right represents the real (measured) makespan in seconds, while the stacked bar on the left represents the makespan computed using the proposed model for GATE with checkpointing. The four stacked elements correspond to the four terms in Eq. (3). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Experiment results—with checkpointing (Exp 3).

Run	ρ	$F(c)$	E_L (s)	m (s)	M real (s)	M model (s)	Model error (%)
30 min #1	0.261	0.239	1988	3060	21 803	23 968	−9.7
30 min #2	0.239	0.211	1026	7680	25 214	27 372	−9.1
30 min #3	0.202	0.184	3643	2640	23 470	23 794	−0.7
30 min mean	–	–	–	4460	23 495	25 024	–
60 min #1	0.183	0.177	1343	1080	20 056	20 744	−3.5
60 min #2	0.196	0.164	2501	840	24 163	21 724	10.6
60 min #3	0.206	0.193	1453	1860	22 060	23 584	−6.2
60 min mean	–	–	–	1260	22 093	21 972	–
120 min #1	0.120	0.100	4319	720	26 071	27 872	−6.4
120 min #2	0.263	0.241	4879	900	26 368	25 813	1.6
120 min #3	0.279	0.250	5336	900	28 378	27 241	3.9
120 min mean	–	–	–	840	26 939	26 990	–

The checkpointing influence is closely related to the failure distribution. Fig. 8 plots the time-to-failure distribution $F(t)$ for the 9 repetitions in this experiment. F is estimated as follows:

$$F(t) = \frac{\text{failed jobs of duration } < t}{\text{failed jobs of duration } < t + \text{ jobs of duration } > t}. \quad (4)$$

Note that jobs that successfully completed before time t are not taken into account as they bring no information about the probability to run longer than time t. When t increases, $F(t)$ is therefore overestimated due to the ignored completed jobs. To correct for that, we use $\tilde{F}$:

$$\tilde{F}(t) = \min(F(t), \rho).$$

This estimation is only valid for $t < M$, and it cannot be extrapolated to longer runs.

These curves all exhibit a similar pattern: most failures occur at the very beginning of the simulation, i.e. before the first checkpoint. This explains why checkpointing has a limited impact in this experiment. Conversely, as shown in Exp 1, the contribution of checkpointing is much more important for longer simulations.

We also noticed that experiments with checkpointing have a rather high failure rate among the merging jobs. Indeed, sites often kill jobs consuming little CPU, which is the case of merge jobs when they are waiting for new results checkpointed by simulation jobs.

The data footprint should also be taken into account. The workflow without checkpointing generates only a few extra files (corresponding to the partial results produced by the multiple mergers), the total number of files being close to 300. The workflow with checkpointing generates an important number of partial results, varying from a minimum of 897 files (with a checkpoint frequency of two hours) to a maximum of 3359 files (with a checkpoint frequency of 30 min).

5. Conclusion

This paper presented an end-to-end framework for executing Monte Carlo simulations on heterogeneous distributed systems.

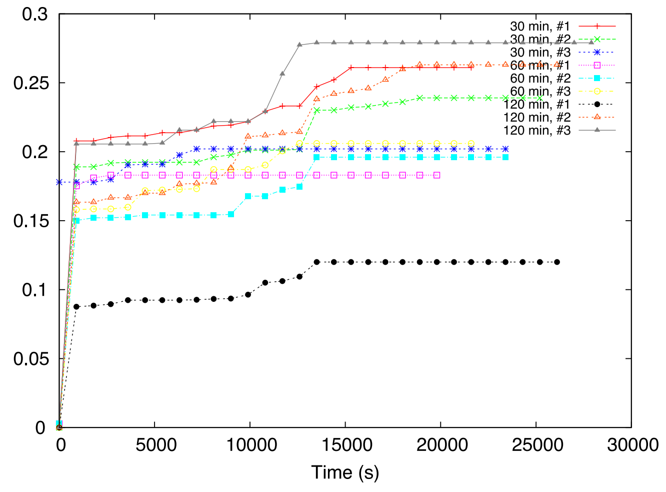


Fig. 8. Measured TTF cumulative distribution on the 9 executions with checkpointing.

Monte Carlo simulations are known to be easily parallelized, but difficulties such as load-balancing on heterogeneous and unreliable resources, as well as merging partial results are not completely overcome. In particular, the merging step can represent a significant amount of additional time in production conditions.

To address these problems, we proposed an approach using a dynamic load-balancing algorithm with multiple parallel mergers. Checkpointing was also proposed to improve reliability, and to enable incremental results merging from partial results.

Three experiments have been conducted on a production infrastructure. The first experiment highlights the necessity of using checkpointing for long simulations. The second one, with different numbers of parallel mergers, shows that using a unique merger is clearly sub-optimal and that the merging time can be reduced from 50% to less than 15% of the total makespan when using multiple parallel mergers. This corresponds to an average makespan decrease of approximately 40% when using 10 parallel mergers. The third experiment, with different checkpointing periods, shows that the checkpointing can be used without penalizing the makespan. Consequently, from an application point of view and with a proper checkpointing frequency, checkpointing could be activated for all simulations: long simulations as presented in Exp 1 would greatly benefit from it, while shorter ones as presented in Exp 2 and 3 would not be penalized.

A model was proposed to explain the measures made in production. It extends previous models by integrating the job failure rate, the merging time and the checkpointing frequency for the workflow with checkpointing. The model is not yet predictive, but it gives a good interpretation of the parameters influencing the makespan. Experimental results fit the model with a relative error of less than 10%. As future work on this topic, we plan to design a model for the merging time and to enrich the global model by taking into account job resubmissions.

All the results were obtained in production conditions, on the European Grid Infrastructure. While experimenting in production ensures that all the assumptions are realistic, it also limits reproducibility and statistical significance. To cope with this issue, we plan to use the experimental data presented in this study to simulate our framework based on toolboxes such as [23,24]. This requires (i) implementing our workflows in the simulation environment, (ii) parameterizing the simulation using traces captured from our real experiments (e.g. job waiting times, fault distributions), and (iii) validating performance results obtained in simulation against real ones. Such a simulator would also allow us to evaluate the model in a wider range of execution conditions.

Our framework can apply to a whole range of Monte Carlo simulators. For the experiments presented here we chose a GATE proton therapy simulation, but GATE is used for a variety of nuclear medicine simulations, such as radiotherapy, emission tomography (positron emission tomography—PET and single photon emission computed tomography—SPECT) and computed tomography (CT). Other Monte Carlo simulators could also use our framework. For instance, experiments were already conducted with a Diffusion Weighted Imaging (DWI) simulator [25].

Based on the conclusions presented in this paper, checkpointing for the GATE application will soon be deployed on the VIP/GATE-Lab portal. The GATE workflow with multiple mergers is already in production and executed by several users daily.

Acknowledgments

This work is co-funded by the French national research agency (ANR), hGATE project under contract no. ANR-09-COSI-004-01. It also falls into the scope of the scientific topics of the French National Grid Institute (IdG). The authors would like to thank the site administrators of the European Grid Initiative and the GGUS support for their work.

References

[1] M. Mascagni, Y. Li, Computational infrastructure for parallel, distributed, and grid-based Monte-Carlo computations, in: LSSC, 2003, pp. 39–52.

[2] L. Maigne, D. Hill, P. Calvat, V. Breton, D. Lazaro, R. Reuillon, Y. Legré, D. Donnarieix, Parallelization of Monte-Carlo simulations and submission to a grid environment, in: Parallel Processing Letters HealthGRID 2004, Vol. 14, Clermont-Ferrand France, 2004, pp. 177–196.

[3] J. Basney, R. Raman, M. Livny, High throughput Monte-Carlo., in: PPSC, 1999.

[4] N. Jacq, J. Salzemann, F. Jacq, Y. Legré, E. Medernach, J. Montagnat, A. Maass, M. Reichstadt, H. Schwichtenberg, M. Sridhar, V. Kasam, M. Zimmermann, M. Hofmann, V. Breton, Grid enabled virtual screening against malaria, Journal of Grid Computing 6 (2008) 29–43.

[5] J.S. Rosenthal, Parallel computing and Monte-Carlo algorithms, Far East Journal of Theoretical Statistics 4 (1999) 207–236.

[6] Y.P. Galyuk, V. Memnonov, S.E. Zhuravleva, V.I. Zolotarev, Grid technology with dynamic load balancing for Monte Carlo simulations, in: 6th International Conference on Applied Parallel Computing Advanced Scientific Computing, Springer-Verlag, London, UK, 2002, pp. 515–520.

[7] R. Procassini, M. O'Brien, J. Taylor, Load Balancing of Parallel Monte-Carlo Transport Calculations, in: Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, Palais des Papes, Avignon, Fra, 2005.

[8] W. Cirne, F. Brasileiro, D. Paranhos, L. Goes, W. Voorsluys, On the efficacy, efficiency and emergent behavior of task replication in large distributed systems, Parallel Computing 33 (2007) 213–234.

[9] S. Camarasu-Pop, T. Glatard, J.T. Moscicki, H. Benoit-Cattin, D. Sarrut, Dynamic partitioning of GATE Monte-Carlo simulations on EGEE, Journal of Grid Computing 8 (2) (2010) 241–259.

[10] J. Mocicki, M. Lamanna, M. Bubak, P. Sloot, Processing moldable tasks on the grid: late job binding with lightweight user-level overlay, Future Generation Computer Systems 27 (6) (2011) 725–736.

[11] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.

[12] Z. Fadika, E. Dede, J. Hartog, M. Govindaraju, MARLA: MapReduce for heterogeneous clusters, CCGrid, in: IEEE International Symposium on Cluster, Cloud and Grid Computing 0, 2012, pp. 49–56.

[13] Z. Guo, G. Fox, M. Zhou, Investigation of data locality in MapReduce, CCGrid, in: IEEE International Symposium on Cluster, Cloud and Grid Computing 0, 2012, pp. 419–426.

[14] J. Ekanayake, S. Pallickara, MapReduce for data intensive scientific analysis, in: Fourth IEEE International Conference on eScience, 2008, pp. 277–284.

[15] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, R. Sears, MapReduce online., in: NSDI, USENIX Association, 2010, pp. 313–328.

[16] J. Plank, An overview of checkpointing in uniprocessor and distributed systems, focusing on implementation and performance, Tech. rep., 1997.

[17] S. Jan, D. Benoit, E. Becheva, T. Carlier, F. Cassol, P. Descourt, T. Frisson, L. Grevillot, L. Guigues, L. Maigne, C. Morel, Y. Perrot, N. Rehfeld, D. Sarrut, D.R. Schaart, S. Stute, U. Pietrzyk, D. Visvikis, N. Zahra, I. Buvat, Gate v6: a major enhancement of the gate simulation platform enabling modelling of ct and radiotherapy, Physics in Medicine and Biology 56 (4) (2011) 881–901.

[18] R. Tudoran, A. Costan, G. Antoniu, H. Soncu, Tomusblobs: Towards communication-efficient storage for MapReduce applications in Azure,

CCGrid, IEEE International Symposium on Cluster, Cloud and Grid Computing 0, 2012, pp. 427–434.

[19] T. Glatard, J. Montagnat, D. Lingrand, X. Pennec, Flexible and efficient workflow deployment of data-intensive applications on grids with MOTEUR, International Journal of High Performance Computing Applications (IJHPCA) 22 (3) (2008) 347–360.

[20] R. Ferreira da Silva, S. Camarasu-Pop, B. Grenier, V. Hamar, D. Manset, J. Montagnat, J. Revillard, J.R. Balderrama, A. Tsaregorodtsev, T. Glatard, Multi-infrastructure workflow execution for medical simulation in the virtual imaging platform, in: HealthGrid 2011, Bristol, UK, 2011.

[21] A. Tsaregorodtsev, M. Bargiotti, N. Brook, A.C. Ramo, G. Castellani, P. Charpentier, C. Cioffi, J. Closier, R.G. Diaz, G. Kuznetsov, Y.Y. Li, R. Nandakumar, S. Paterson, R. Santinelli, A.C. Smith, M.S. Miguelez, S.G. Jimenez, Dirac: a community grid solution, Journal of Physics: Conference Series 119 (6) (2008).

[22] L. Grevillot, D. Bertrand, F. Dessy, N. Freud, D. Sarrut, A Monte Carlo pencil beam scanning model for proton treatment plan simulation using GATE/GEANT4, Physics in Medicine and Biology 56 (16) (2011) 5203–5219.

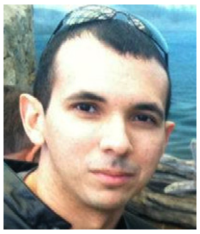
[23] H. Casanova, A. Legrand, M. Quinson, SimGrid: a Generic Framework for Large-Scale Distributed Experiments, in: 10th IEEE International Conference on Computer Modeling and Simulation, 2008.

[24] R. Buyya, M.M. Murshed, Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience (CCPE) 14 (13–15) (2002) 1175–1220.

[25] L. Wang, Y. Zhu, H. Li, W. Liu, I.E. Magnin, Multiscale modeling and simulation of the cardiac fiber architecture for dmri, IEEE Transactions on Biomedical Engineering 59 (1) (2012) 16–19.

Sorina Camarasu-Pop received her Engineering degree in telecommunications in 2007 and her Master's degree in 2008, both from the National Institute for Applied Sciences of Lyon (INSA-Lyon, France). Since 2007 she is a CNRS research engineer at Creatis in Lyon, France. Her activity is focused on porting and optimizing the execution of medical image processing applications on heterogeneous distributed systems.

Tristan Glatard obtained a Ph.D. in grid computing applied to medical image analysis from the University of Nice Sophia-Antipolis, France in 2007. He spent a year as a post-doc at the University of Amsterdam working in the medical group of the Virtual-Lab for e-Science project. He is now a researcher at CNRS Creatis in Lyon, France.



Rafael Ferreira da Silva received his Master's degree in computer science from Universidade Federal de Campina Grande, Brazil, in 2010. He is currently a CNRS software engineer at Creatis in Lyon, France, where he is conducting his Ph.D. studies in computer science. His research is focused on workflow execution service on heterogeneous distributed systems.

Pierre Gueth graduated from ENS Cachan in 2005 with a bachelor's degree in applied physics. He received his master's degree in signal and image processing from INSA in 2006. In his thesis, defended in 2001 at CREATIS-LRMN, Lyon, he studied beamforming and displacement estimation for medical ultrasound. In his current postdoctorate position in the OpenGate collaboration, he develops the GATE software and study image guided hadrontheraphy using prompt gamma.

David Sarrut is a CNRS researcher (CR1) working at CREATIS in the fields of medical image processing and digital simulations for radiation cancer therapy. His research interests concern Image-Guided Radiation Therapy (IGRT), as well as Monte Carlo simulations of radiation therapy treatments, both with conventional photon devices and with hadron therapy. He initiated a continuous collaboration with the Léon Bérard cancer center and animates a research group localised within this hospital. He is an active member of the OpenGate collaboration and initiated the extension of Gate to radiation therapy applications. He supervised 9 defended Ph.D. thesis, and is currently supervisor of 3 Ph.D. students. He has about 30 co-authored journal publications and 45 conference proceedings. His main publications regarding simulations are in PMB and Medical Physics. Web site: http://www.creatis.insalyon.fr/~dsarrut.

Hugues Benoit-Cattin received in 1992 the Engineer degree (electrical engineering) and in 1995 the Ph.D degree (wavelet image coding of medical images), both from INSA Lyon, France. He is Professor at INSA Lyon, at the Telecommunications Department. His teaching activities mainly concern information theory, signal and image processing. Since 1 March 2008, he is the Dean of the INSA Lyon Telecoms Department. A member of the Models and Images Team at CREATIS, his research interests include image segmentation as well as MRI image analysis and simulation.