THÈSE

présentée à

L'Institut National des Sciences Appliquées de Lyon
Ecole Doctorale Electronique, Electrotechnique et Automatique

pour obtenir le titre de

Docteur

spécialité

Informatique

Présentée par

Sorina CAMARASU-POP

Exploitation d'infrastructures hétérogènes de calcul distribué pour la simulation Monte-Carlo dans le domaine médical

Jury

September 13, 2013

Abstract

Particle-tracking Monte-Carlo applications are easily parallelizable, but efficient parallelization on computing grids is difficult to achieve. Advanced scheduling strategies and parallelization methods are required to cope with failures and resource heterogeneity on distributed architectures. Moreover, the merging of partial simulation results is also a critical step. In this context, the main goal of our work is to propose new strategies for a faster and more reliable execution of Monte-Carlo applications on computing grids. These strategies concern both the computing and merging phases of Monte-Carlo applications and aim at being used in production.

In this thesis, we introduce a parallelization approach based on pilots jobs and on a new dynamic partitioning algorithm. Results obtained on the production European Grid Infrastructure (EGI) using the GATE application show that pilot jobs bring strong improvement w.r.t. regular metascheduling and that the proposed dynamic partitioning algorithm solves the load-balancing problem of particle-tracking Monte-Carlo applications executed in parallel on distributed heterogeneous systems. Since all tasks complete almost simultaneously, our method can be considered optimal both in terms of resource usage and makespan.

We also propose advanced merging strategies with multiple parallel mergers. Checkpointing is used to enable incremental result merging from partial results and to improve reliability. A model is proposed to analyze the behavior of the complete framework and help tune its parameters. Experimental results 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.

To evaluate our load balancing and merging strategies, we implement an end-to-end SimGrid-based simulation of the previously described framework for Monte-Carlo computations on EGI. Simulated and real makespans are consistent, and conclusions drawn in production about the influence of application parameters such as the checkpointing frequency and the number of mergers are also made in simulation. These results open the door to better and faster experimentation.

To illustrate the outcome of the proposed framework, we present some usage statistics and a few examples of results obtained in production. These results show that our experience in production is significant in terms of users and executions, that the dynamic load balancing can be used extensively in production, and that it significantly improves performance regardless of the variable grid conditions.

Remerciements

Cette thèse n'aurait jamais vu le jour sans le concours de plusieurs personnes que je tiens à remercier chaleureusement.

Je tiens à remercier tout particulièrement mes directeurs de thèse, Hugues Benoit-Cattin et Tristan Glatard, sans lesquels je ne me serais sans doute pas lancée dans cette aventure. Au delà de leur encadrement sans faille, ce sont deux personnes exceptionnelles avec lesquelles je suis très heureuse d'avoir la chance de travailler.

Merci à Hugues pour ses conseils toujours très avisés, sa capacité à prendre du recul et à proposer de nouvelles pistes de réflexion, ainsi que pour la confiance qu'il m'a toujours inspirée.

Merci à Tristan pour les échanges quotidiens, ses précieux conseils scientifiques et techniques, sa disponibilité, son dynamisme et son enthousiasme contagieux.

Je tiens aussi à remercier Isabelle Magnin, directrice du laboratoire Creatis, qui m'a soutenu et encouragé dans mon activité et qui a donné son accord pour que je puisse réaliser cette thèse tout en étant ingénieur de recherche au laboratoire.

Je remercie les membres de mon jury, Johan Montagnat, Jakub Mościcki, Arnaud Legrand, Frédéric Desprez, Ignacio Blanquer et David Sarrut, qui, pour la plupart, ont aussi participé à mon comité de thèse. Merci pour leur disponibilité et pour leurs remarques fructueuses.

Many thanks in particular to Ignacio Blanquer and Frédéric Desprez for having accepted the time-consuming task of reviewing this manuscript. I hope you will enjoy reading it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Monte-Carlo simulations are used in several scientific domains to produce realistic results from repeated sampling of events generated by pseudo-random algorithms. The Monte Carlo method was defined by [Halton, 1970] as "representing the solution of a problem as a parameter of a hypothetical population, and using a random sequence of numbers to construct a sample of the population, from which statistical estimates of the parameter can be obtained". Figure 1.1 illustrates a classical example of how the Monte-Carlo method can be applied to approximate the value of π. Given a circle inscribed in a unit square, the method consists in (i) scattering uniformly some objects of uniform size over the square and (ii) counting the number of objects inside the circle and the total number of objects inside the square. The ratio of the two counts is an estimate of the ratio of the two areas, i.e. $\pi/4$.

In the field of medical imaging and radiotherapy, GATE (Geant4 Application for Emission Tomography) [Allison et al., 2006] is an advanced open source particle-tracking

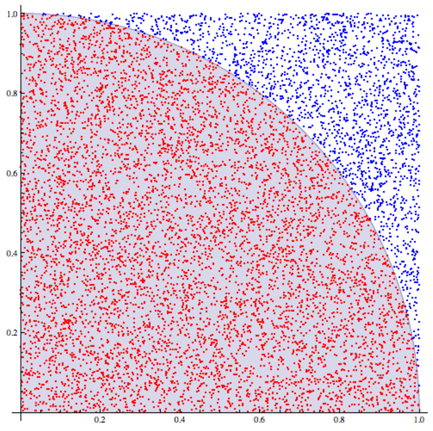


FIGURE 1.1: Monte Carlo method applied to approximating the value of π. After placing 30000 random points, the estimate for π is within 0.07% of the actual value. Credits: Wikipedia, the free encyclopedia

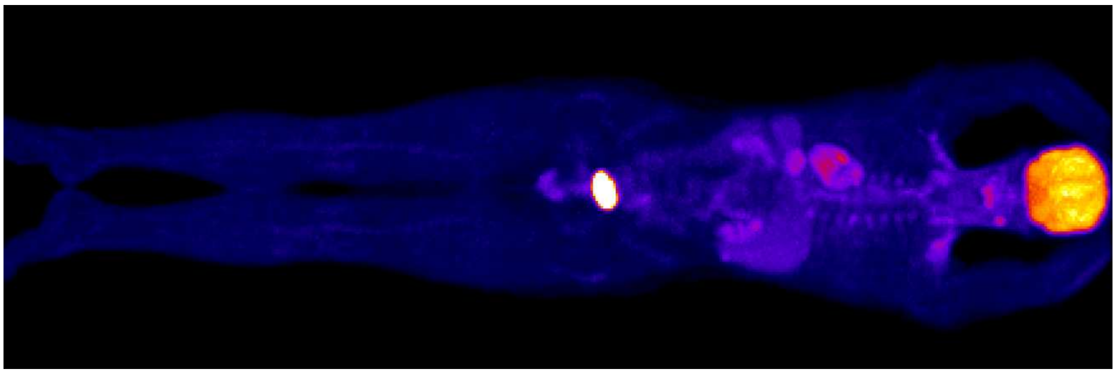


FIGURE 1.2: 3D whole-body F18-FDG PET scan simulated with GATE, representing 4,000 CPU hours (5.3 months). Credits: IMNC-IN2P3 (CNRS UMR 8165).

Monte-Carlo simulator developed by the international OpenGATE[1] collaboration. The simulation in a particle tracking Monte-Carlo system consists in the successive stochastic tracking through matter of a large number of individual particles. Each particle has an initial set of properties (type, location, direction, energy, etc) and its interaction with matter is determined according to realistic interaction probabilities and angular distributions. GATE currently supports simulations of Emission Tomography (Positron Emission Tomography - PET and Single Photon Emission Computed Tomography - SPECT), Computed Tomography (CT) and Radiotherapy experiments. Used by a large international community, GATE now plays a key role in the design of new medical imaging devices, in the optimization of acquisition protocols and in the development and assessment of image reconstruction algorithms and correction techniques. It can also be used for dose calculation in radiotherapy experiments.

To produce accurate results, Monte-Carlo simulations require a large number of statistically independent events, which has a strong impact on the computing time of the simulation. As an example, Figure 1.2 illustrates a 3D whole-body F18-FDG PET scan simulated with GATE and representing 4,000 CPU hours (5.3 months). To cope with such a computing challenge, Monte-Carlo simulations are commonly parallelized using a split and merge pattern. They can thus exploit important amounts of distributed resources available world-wide.

Distributed computing is a widely-used processing method where different parts of a program are processed simultaneously on two or more computers that are communicating with each other over a network. Distributed computing originated with the creation of clusters, which were then interconnected into grids, and it continues to evolve with the more recent cloud technology. While cluster nodes are connected by a local area network (LAN), grids and clouds are geographically distributed, usually covering multiple administrative domains. A detailed comparison between these technologies is available

1. http://www.opengatecollaboration.org

in [Sadashiv and Kumar, 2011]. Grid technologies are mainly used by High Throughput Computing (HTC) systems, which focus on the efficient execution of a large number of loosely-coupled tasks, as opposed to High Performance Computing (HPC) systems, which focus on the efficient execution of tightly-coupled tasks. In this thesis, we will refer mostly to grid computing and consequently, to HTC systems.

Various strategies have been proposed to speed-up the execution of Monte-Carlo simulations on distributed platforms [Maigne et al., 2004, Stokes-Ress et al., 2009]. Nevertheless, they mostly use static splitting and are not well suited to grids, where resources are heterogeneous and unreliable. Moreover, 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 (the execution time as perceived by the user).

High failure rates have strong consequences on the performance of applications. Failure rates of more than 10% are commonly observed on distributed production infrastructures such as EGI [Jacq et al., 2008]. Consequently, fault tolerance [Garg and Singh, 2011] becomes essential in grid computing. Commonly utilized techniques for providing fault tolerance are pilot jobs, job checkpointing and task replication. Pilot jobs submit generic jobs that retrieve and execute user tasks as soon as they get running on the available resources. Thus, fault-tolerance can be achieved at two levels: first, pilot jobs check their execution environment before retrieving user jobs, which reduces the failure rate of user jobs; second, in order to prevent recurrent failures, faulty pilots are removed and failed tasks are pulled by a different pilot. 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. Task replication consists of dispatching multiple replicas of a task and using the result from the first replica to finish. It can achieve very good fault-tolerance, but replicated tasks waste computing cycles that could be used for other computations.

The performance of strategies, e.g. scheduling algorithms, used on distributed computing infrastructures is difficult to assess in production due to the variable execution conditions across experiments. Evaluation methods can combine multiple approaches, such as modeling, experimentation and simulation. Modeling can help understand the behavior of distributed applications and tune parameters such as the checkpointing delay. Nevertheless, it remains 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. At the same time, production experiments ensure realistic conditions and results, but are cumbersome and difficult to reproduce. Reproducing production experiments in simulation becomes thus interesting, since it enables faster experimentation and controlled conditions.

In this context, the main challenge addressed by this thesis consists in finding new strategies for a faster and more reliable execution of Monte-Carlo computations on heterogeneous distributed infrastructures. These strategies concern both the computing and merging phases of Monte-Carlo applications and aim at being used in production. Last but not least, they are validated using multiple approaches including experimentation, modeling and simulation.

This manuscript is organized as follows :

Chapter 2 presents related work in the fields of distributed computing, Monte-Carlo simulation on grids and validation strategies. It gives a short overview of some of the existing solutions and underlines the need of new strategies able to address the challenges mentioned in the previous paragraph.

Chapter 3 deals with the issue of efficient parallelization of Monte-Carlo simulations on distributed heterogeneous platforms by introducing a new dynamic partitioning algorithm. First, we present a pilot-job strategy and its comparison to regular metasheduling. Then, we introduce our new dynamic load-balancing algorithm and its implementation using pilot jobs. The proposed algorithm is evaluated in production conditions on EGI.

Chapter 4 deals with the merging phase, which consists in downloading and merging all partial results into one final output. Depending on the number of partial results, their availability and transfer times, the merging phase can significantly increase the application makespan. In order to cope with this issue, we propose advanced merging strategies with multiple parallel mergers. Checkpointing is also used to enable incremental result merging from partial results and to improve reliability. The proposed strategies are evaluated in production conditions. A model aiming at explaining measures made in production is also introduced.

Chapter 5 presents an end-to-end SimGrid-based simulation [Casanova et al., 2008] of the previously described strategies integrated into a complete framework for Monte-Carlo computations on EGI. Middleware services are simulated by SimGrid processes, while the deployment of pilot jobs is simulated by a random selection of platform hosts, and a matching of their latencies and failure times with real traces. The Monte-Carlo application workflow is calibrated to address performance discrepancies between the real and simulated network and CPUs. The simulation is evaluated against real executions of the GATE Monte-Carlo application.

Chapter 6 gives some usage statistics and a few examples of results obtained in production using the GATE-Lab, a tool that we developed and that has both motivated the previously presented contributions and helped validate them in a production environment, beyond controlled experiments. The GATE-Lab counts today 300 users and completes more than 150 GATE executions per month. Results show that the proposed dynamic load balancing is extensively used in production and that it significantly improves performance.

Chapter 7 concludes the manuscript with a general discussion and perspectives.

Finally, Chapter 8 is a summary of the thesis in French.

Chapter 2

State of the art: execution and simulation of Monte-Carlo computations on distributed architectures

Abstract *This chapter presents related work and current status of the advances in the fields of distributed computing, Monte-Carlo simulation on grids and validation strategies. It gives a short overview of some of the existing grid computing infrastructures, jobs distributing strategies, application description, fault tolerance, parallelization methods for Monte-Carlo applications, as well as evaluation approaches including experimentation in production and controlled environments, simulation and formal modeling. The chapter concludes by underlying the need for new strategies able to provide faster and more reliable executions of Monte-Carlo computations on heterogeneous distributed infrastructures.*

2.1 Introduction

Grid technologies are complex systems that have to deal with the challenges of (i) job distribution strategies, (ii) application parallelization, and (iii) reliability and fault-tolerance. These challenges will be discussed in Section 2.2.

The important computing needs of Monte-Carlo simulations, as well as their facility to be split into loosely-coupled tasks, have encouraged their porting to distributed infrastructures from their early beginning [Rosenthal, 1999]. Meanwhile, various strategies have been proposed to efficiently execute Monte-Carlo simulations on distributed platforms.

An overview of these strategies, as well as of the science gateways allowing to execute Monte-Carlo applications on distributed infrastructures, is given in Section 2.3.

Validation strategies combining multiple approaches are important in order to assess the proposed methods. In the field of distributed computing, these approaches can vary from experiments performed on the target production infrastructure to experiments in controlled environments, simulation and even formal modeling. In many cases, simulation offers an attractive alternative to experimentation because simulation experiments are fully repeatable and configurable. Nevertheless, their realism can be subject to discussions. Examples of validation strategies are presented and discussed in Section 2.4.

2.2 Distributed computing: infrastructures and application deployment

This section introduces some of the existing grid infrastructures and their challenges. First, these large infrastructures require job distribution strategies to dispatch the load among the multiple heterogeneous grid sites. Second, on top of these distribution strategies, applications have to be able to take advantage of the multiple resources relying on parallelism. For some types of applications this can be achieved by using a parallel meta-language description. Last but not least, these complex infrastructures are prone to errors which prevent applications from finishing their execution. Fault tolerance strategies are then needed in order to satisfy user requirements.

As explained in chapter 1, in the following and throughout this work, distributed computing mostly refers to grid computing.

2.2.1 Grid computing infrastructures

Grid computing originated in academia in the mid 1990s and aimed at aggregating the power of widely distributed resources in order to provide non-trivial services to users. Meanwhile, grids have become an effective way for resource sharing through multi-institutional virtual organizations (VOs) [Foster et al., 2001].

Figure 2.1 illustrates the grid taxonomy proposed by [Krauter et al., 2002], which classifies grids according to their resource models: computational, data or service grids. Nowadays grids tend to offer simultaneously these three types of resources and differ mostly in size and purpose. We can thus differentiate (i) production grids (e.g. EGI, NorduGrid, PRACE, OSG, TeraGrid/XSEDE), which are large infrastructures providing resources

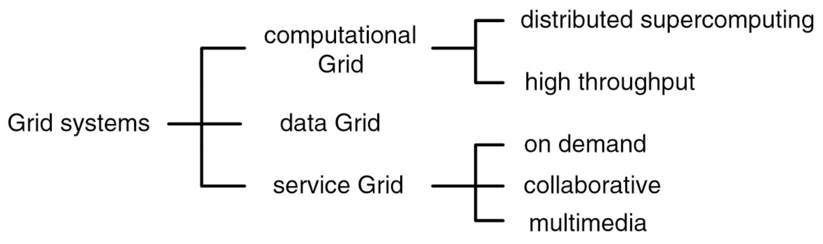


FIGURE 2.1: Grid taxonomy extracted from [Krauter et al., 2002].

for production purposes and (ii) experimental grids (e.g. Grid5000, FutureGrid), which are testbeds designed to support reproducible experiments in academia.

The largest production grids in Europe are EGI, NorduGrid and PRACE. The European Grid Infrastructure[1] (EGI) is the follower of the Enabling Grids for E-sciencE (EGEE) project, which was based in the European Union and included sites in Asia and the United States. EGEE, along with the Large Hadron Collider (LHC) Computing Grid (LCG), was developed to support production experiments using the CERN LHC. EGI currently hosts more than 200 VOs for communities with interests as diverse as Earth Sciences, Computer Sciences and Mathematics, Fusion, Life Sciences or High-Energy Physics. NorduGrid[2] aims at delivering a robust, scalable, portable and fully featured solution for a global computational and data Grid infrastructure suitable for production-level research tasks. It was originally established by several North European academic and research organisations, but the new Collaboration Agreement signed in 2011 has 11 partners from 10 countries. PRACE[3] (Partnership for Advanced Computing in Europe) aims at developing a distributed high performance computing (HPC) infrastructure based on the national supercomputing centers in Europe.

In the United States of America, the Extreme Science and Engineering Digital Environment[4] (XSEDE) replaces and expands on the former TeraGrid infrastructure. TeraGrid was a National Science Foundation (NSF) cyberinfrastructure project coordinated through the Grid Infrastructure Group (GIG) at the University of Chicago and combining resources at eleven partner sites. The TeraGrid integrated high-performance computers, data resources and tools, and experimental facilities. The Open Science Grid[5] (OSG) is a multi-disciplinary partnership to federate local, regional, community and national cyberinfrastructures to meet the needs of research and academic communities at all scales. Today, the OSG community brings together over 100 sites that provide computational and storage resources.

1. http://www.egi.eu
2. http://www.nordugrid.org/
3. https://www.surfsara.nl/project/prace
4. https://www.xsede.org/home
5. https://www.opensciencegrid.org

Desktop grids are a special kind of production grids, known as volunteer computing or Public-Resource Computing systems, since they gather resources contributed by volunteers all over the world. Resources are often personal computers, but also sometimes clusters, controlled by their owners (e.g., universities). One of the first and most well-known desktop grid is SETI@home [6][Anderson et al., 2002], which aims at using Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). According to [Balaton et al., 2008], desktop grids follow a pull model, as opposed to service grids which follow a push model. [Balaton et al., 2008] also classify desktop grids into two categories: public or global and non-public or local.

Among experimental grids, we can mention DAS4 [7][Bal et al., 2000] and Grid5000 [8][Cappello and Bal, 2007] in Europe and FutureGrid [9] in the USA. DAS-4 (the Distributed ASCI Supercomputer 4) is a six-cluster wide-area distributed system designed by the Advanced School for Computing and Imaging (ASCI) in the Netherlands. The goal of DAS-4 is to provide a common computational infrastructure for researchers within ASCI, who work on various aspects of parallel, distributed, grid and cloud computing, and large-scale multimedia content analysis. Grid5000 is a scientific instrument designed to support experiment-driven research in all areas of computer science related to parallel, large-scale or distributed computing and networking. It aims at providing a highly reconfigurable, controllable and monitorable experimental platform to its users. FutureGrid is an NSF-funded test-bed that contains geographically distributed resources and services. While other XSEDE systems support production computational science, FutureGrid supports experimental computer science research, software evaluation and testing, education and training, as well as the development of computational science applications.

2.2.2 Job distribution strategies

All grid infrastructures require job distribution strategies able to dispatch the load among their multiple heterogeneous sites. In the following we will review some of the existing grid meta-schedulers and pilot-job systems.

2.2.2.1 Grid meta-schedulers

In a grid environment, meta-scheduling is the process of distributing jobs to the grid sites [Talia et al., 2008], which are then responsible for scheduling tasks on their own worker nodes. [Krauter et al., 2002] propose a taxonomy of grid resource management

6. http://setiathome.berkeley.edu/index.php
7. http://www.cs.vu.nl/das4
8. https://www.grid5000.fr
9. https://www.xsede.org/futuregrid

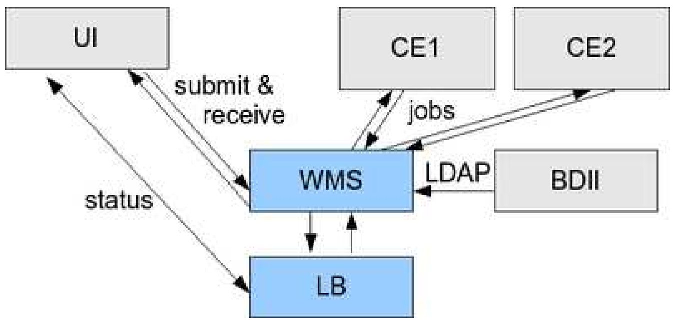


FIGURE 2.2: gLite scheduling mechanism. Jobs submitted through the User Interface (UI) are taken into account by the WMS, which dispatches them to Computing Elements (CE) using the Berkeley Database Information Index (BDII). Job statuses are available through the Logging and Bookkeeping (LB) system.

systems for distributed computing and classifies meta-schedulers in three categories : (i) centralized, (ii) hierarchical and (iii) decentralized.

In the centralized organization, there is only one scheduling controller that is responsible for the system-wide decision making. The advantages of this kind of organization include easy management, simple deployment and the ability to co-allocate resources. The disadvantages are mainly the lack of scalability and the lack of fault-tolerance, since the central meta-scheduler can quickly become a bottleneck and since it represents a single point of failure. One example of centralized scheduling is the gLite [Laure et al., 2006] Workload Management System (WMS) used in EGI. Figure 2.2 illustrates the gLite scheduling mechanism. Jobs submitted by the user through the User Interface (UI) are taken into account by the WMS, which queues the user requests and dispatches them to the different computing centers available. The gateway to each computing center is one or more Computing Element (CE) that will distribute the workload over the Worker Nodes (WN) available at this center. The WMS uses the Berkeley Database Information Index (BDII) to retrieve information about the status and performance of the resources at the VO level. Job statuses are available through the Logging and Bookkeeping (LB) system, which tracks jobs in terms of events (important points of job life, e.g. submission, finding a matching CE, starting execution etc.).

Hierarchical scheduling addresses the scalability and fault-tolerance issues of the centralized scheduling, but still has single points of failure and is more difficult to administrate. In the Distributed Interactive Engineering Toolbox (DIET) [Dail and Desprez, 2006] middleware, a distributed hierarchy of agents divides up the work of scheduling tasks. The servers themselves are responsible for collecting and storing their own resource information and for making performance prediction. When a Master Agent (MA) receives a client request, it (1) verifies that the service requested exists in the hierarchy, (2) collects a list of its children that are thought to offer the service, and (3) forwards the request to those subtrees. DIET returns to the client program multiple server choices, sorted in order of the predicted desirability of the servers. The client finally selects a server from the list.

Decentralized systems address several important issues such as fault-tolerance and scalability, but also introduces several problems such as management, usage tracking, and co-allocation. ARC [Elmroth and Gardfjall, 2005] is a decentralized scheduler used by NorduGrid. ARC enforces locally and globally scoped share policies, allowing local resource capacity as well as global grid capacity to be logically divided across different groups of users. A job that is submitted to the resource is first received by the local workload manager, i.e. the NorduGrid ARC Grid Manager, which may interact with an accounting system before granting the job access. The job is then handed over to the local scheduler, which calculates job priorities.

2.2.2.2 Pilot jobs

Large production grids are often distributed among hundreds of independent sites, rendering the task of meta-schedulers very difficult. On top of grid meta-schedulers, pilot jobs offer a scheduling paradigm trying to cope with the heterogeneity of large-scale production grids and to bring increased reliability. Pilot jobs are generic jobs which are submitted on distributed resources and which, as soon as they start executing, pull user tasks from a common pool of waiting tasks. Upon finishing their current task, pilot jobs will pull a new one as long as the pool is not empty. When there are no waiting tasks left, pilots die in order to free the resource. This pull mechanism is similar to a certain extent to work-stealing scheduling algorithms [Janjic and Hammond, 2010, Quintin and Wagner, 2012] (mostly used for HPC), in which underutilized processors attempt to “steal” tasks from other processors. In the case of pilot jobs, they “steal” tasks from the common pool.

Most big experiments, e.g. ATLAS [10][Maeno et al., 2011] and CMS [11][Fanfani et al., 2010], now rely on pilot jobs. Several pilot-job frameworks have been proposed in the last few years. A few examples are DIANE [Mościcki, 2003], WISDOM [Ahn et al., 2008, Jacq et al., 2008], BOINC [Kacsuk et al., 2008] (mainly used for desktop grids), PanDA [Maeno, 2008], DIRAC [Tsaregorodtsev et al., 2008, Tsaregorodtsev et al., 2009] and GlideInWMS [Sfiligoi, 2008], which is based on the Condor [Tannenbaum et al., 2001, Thain et al., 2005] pilot-job submission framework called Condor GlideIn. In the following, we will shortly present the basic mechanisms and advantages of such systems, taking as example glideinWMS and DIRAC.

10. http://atlas.web.cern.ch/Atlas
11. http://cms.web.cern.ch/

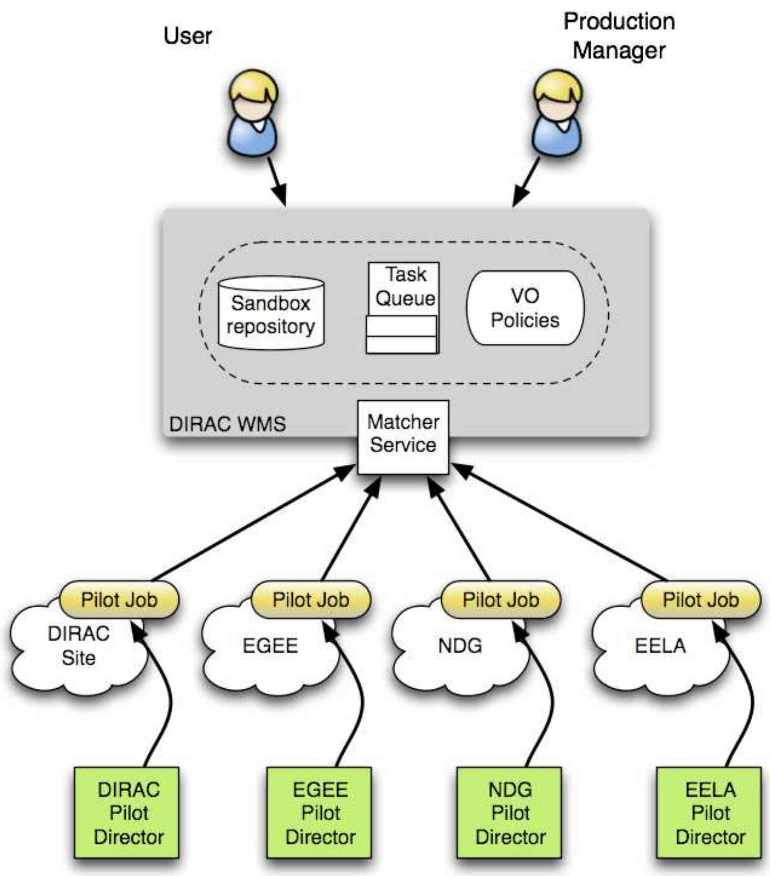


FIGURE 2.3: DIRAC Workload Management with pilot jobs reproduced from [Ferreira da Silva et al., 2011].

The GlideIn Workload Management System (glideinWMS) [Sfiligoi, 2008] has a centralized pilot based submission system called GlideIn Factory (GF). The GlideIn factory is responsible to know which grid sites are available and what are the site attributes, and to advertise these information as "entry-point ClassAds" to a dedicated collector known as the WMS pool. The frontend plays the role of a matchmaker, internally matching user jobs to the entry-point ClassAds, and then requesting the needed amount of pilots, also known as glideins, to the factory. Finally, the factory will submit the glideins that will start the Condor daemons responsible for resource handling. Pilots are sent to all the grid sites that are supposed to be able to run the job. The first pilot that starts gets the job. The remaining glideins will either start another job that can run there (even if they were not submitted for the purpose of serving that job), or terminate within a short period of time. As long as there is a significant amount of jobs in the queue, only a small fraction of pilots will terminate without performing any useful work.

The DIRAC Workload Management System (WMS) is represented on Figure 2.3. All the jobs are submitted to the central Task Queue and pilot jobs are sent to Worker Nodes at the same time. Pilot jobs run special agents that "pull" user jobs from the Task Queue, set up their environment and steer their execution. The centralization of user jobs in the Task Queue enables community policy enforcements while decentralized priority rules defined at the site level are imprecise and suffer scalability problems.

The pilot-job model presents several advantages in terms of latency reduction, load balancing and fault-tolerance [Germain Renaud et al., 2008]. Latency is reduced since pilots, once they get running, will execute multiple tasks one after the other. Load balancing can be achieved by splitting the workload into more tasks than available resources. In this way, the fastest pilots (little queuing time and/or powerful processor) will pull a maximum of tasks from the master pool. Fault-tolerance is achieved at two levels. First, pilot jobs check their execution environment before retrieving user jobs, which reduces the failure rate of user jobs. Second, in order to prevent recurrent failures, faulty pilots are removed and failed tasks are pulled by a different pilot.

Pilot jobs limitations are mainly related to pilot connectivity and data pre-staging. Pilot jobs require outgoing network connectivity from worker nodes, which may not be allowed at all grid sites. Data pre-staging is a functionality allowing to copy input data on the worker node prior to the task execution. This functionality is provided by middlewares such as ARC, where tasks are allocated to resources well before their execution. Pilot jobs only retrieve a task when they become available, being thus unable to pre-stage the files of a future task.

2.2.3 Application description

When using distributed computing, one of the main interests is reducing the application makespan by splitting the workload into multiple tasks that can be executed concurrently. For some types of applications, this can be done by using workflow languages. For applications dealing with data-intensive processing of large datasets, the description can become more specialized using the Map-Reduce model. One specific step of the workflow of some applications is the merging phase.

2.2.3.1 Workflows

Applications executed on large distributed environments are often described with workflows which facilitate application porting and reusability [Kacsuk and Sipos, 2005, Maheshwari et al., 2009, Glatard et al., 2008a, Deelman et al., 2005, Oinn et al., 2004, Fahringer et al., 2004, Taylor et al., 2007, Fahringer et al., 2007, Altintas et al., 2004], and enable parallelism. These workflows are interpreted by engines that generate tasks from application descriptions.

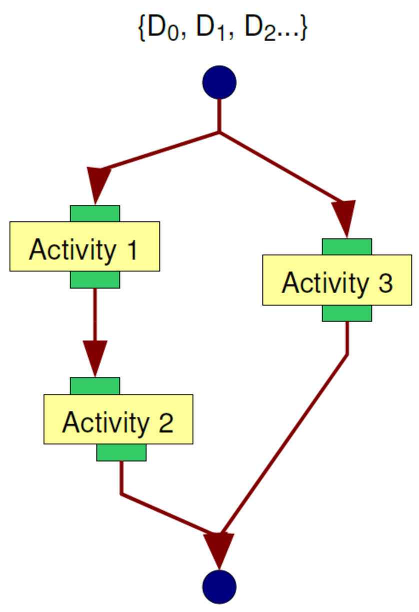


Figure 2.4: Example of a graphical representation of a scientific workflow reproduced from [Montagnat et al., 2009]. Activities (yellow boxes) are interconnected by data links (red arrows), which express data exchanges between successive activities. They also imply time dependencies between the firing of two different components.

By exploiting workflows, users expect to benefit from a parallel implementation without explicitly writing parallel code. In [Pautasso and Alonso, 2006], authors refer to 2 types of parallelism provided by workflow languages : simple parallelism and data parallelism. Simple parallelism denotes that tasks with no dependencies can be executed concurrently. With data parallelism, data is partitioned and the same task is applied in parallel over each (independent) partition. Once all data partitions have been processed, a merging phase may follow in order to aggregate the results. This pattern, inspired from the Single Program Multiple Data (SPMD) streams, is often used to model embarrassingly parallel computations such as parameter-sweep simulations. In many scientific application, data parallelism represents the primary source of performance gain expectation. Especially on large scale distributed systems such as grids, data parallelism is a coarse parallelism that can be efficiently exploited [Montagnat et al., 2009].

Figure 2.4 illustrates an example of a graphical representation of a scientific workflow. Simple parallelism is expressed through activities 1 and 3 which can be scheduled for execution using parallel jobs. Data parallelism is expressed through the multiple input datasets (D0, D1, D2), for which the whole workflow (activities 1, 2 and 3) can be executed in parallel.

2.2.3.2 Map-Reduce

MapReduce [Dean and Ghemawat, 2004] is a programming model and an associated implementation for data-intensive processing of large datasets on distributed platforms like clusters or grids. It has been widely adopted by many business and scientific applications and there are currently increasing efforts for workflows and systems to work with the MapReduce programming model [Crawl et al., 2011]. The basic functioning of the MapReduce implementation is the following:

– In the Map step, the master node partitions the input into smaller sub-problems and distributes them to worker nodes. The worker nodes process these smaller problems, and return their partial results to the master node.
– During the Reduce step, the partial results are combined in order to get the final output. It may happen that there are several reducers, each of them producing one independent result.

One of the first and most well-known implementations is the Google MapReduce library [Dean and Ghemawat, 2008]. The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function. The number of partitions (R) and the partitioning function are specified by the user. After successful completion, the output of the MapReduce execution is available in the R output files (one per reduce task, with file names as specified by the user). Depending on the application, users may not need to combine these R output files into one file.

If a single final result is needed, a combine stage can be added as described in [Ekanayake and Pallickara, 2008], where MapReduce is used for scientific applications (HEP and KMeans clustering). Hadoop [12] (Apache's MapReduce implementation) is mentioned to schedule the MapReduce computation tasks depending on the data locality and hence improving the overall I/O bandwidth. This setup is well suited for an environment where Hadoop is installed in a large cluster of commodity machines. [Ekanayake and Pallickara, 2008] evaluate the total time and speed-up with respect to data size, but no information is given on the performance of the reduce and combine steps individually.

2.2.3.3 Merging

As presented previously, both for the workflow-based data parallelism and for the Map-Reduce approaches, data processed in parallel may need to be merged in order to produce

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

one final result. Nevertheless, most of the existing literature concentrates on the processing itself and says little about the merging of partial results.

In [Condie et al., 2010], authors monitor map and reduce progresses separately showing that the reduce step can take much more time than the map step. [Antoniu et al., 2012] also note that the reduce phase can be costly and proposes a MapIterativeReduce strategy for reduce-intensive applications. [Stokes-Ress et al., 2009] also mention the merging problem for Monte-Carlo simulations on heterogeneous infrastructures.

The merging problem is 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 (or even longer) than the processing phase. The merging phase can thus represent a key element in improving the performance of distributed applications.

2.2.4 Fault tolerance strategies

Large scale computational grid systems, where geographically distributed nodes cooperate to execute the multiple tasks of different applications, are prone to errors. [Ben-Yehuda et al., 2012] mention resource availability averages of 70% or less, while [Li et al., 2010] report an average task error ratio of 7.8%, despite the use of pilot-jobs which intrinsically provide a certain level of fault tolerance. Figure 2.5 illustrates the availability and reliability percentages across EGI as presented by technical reports[13]. In order to cope with this issue and to satisfy user requirements in terms of QoS, fault tolerance has to be provided at the application level. Commonly utilized techniques for providing fault tolerance at the application level are job checkpointing and task replication.

2.2.4.1 Checkpointing

Checkpointing consists in saving the state of a running program so that it can be reconstructed later in time [Plank, 1997]. It has various purposes, among which providing fault-tolerance and accessing partial results. These are valuable for parallel Monte-Carlo applications running on heterogeneous platforms that are prone to failures.

Some grid/cluster computing systems like Condor [Basney et al., 1999] 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,

13. https://documents.egi.eu/public/RetrieveFile

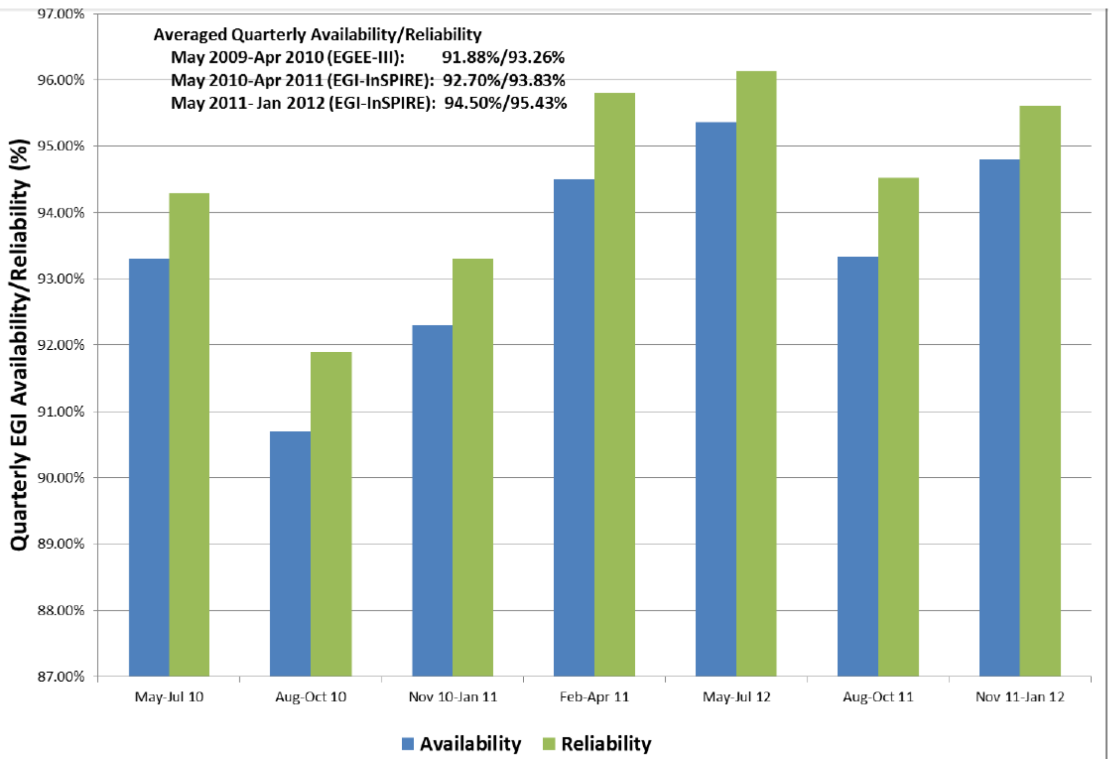


FIGURE 2.5: Availability and reliability percentages across EGI from May 2010 to January 2012 reproduced from EGI technical reports

but they are platform-dependent and may not be suited for large applications requiring significant disk and network resources.

Application-level checkpointing [Mascagni and Li, 2003a] is often more efficient since the application can choose to save the minimal information required to resume computation. As described in [Mascagni and Li, 2003a], Monte-Carlo applications generally need to save only a relatively small amount of information. For instance, the GATE [Jan et al., 2011] 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.

2.2.4.2 Task replication

Task replication consists of dispatching multiple replicas of a task and using the result from the first completed replica [Cirne et al., 2007]. Task replication is able to achieve good performance even in the absence of information on tasks and resources. Nevertheless, replicated tasks waste computing cycles that could be used for other computations.

Task replication is used by a large number of applications [Mascagni and Li, 2003b, Litke et al., 2007, Ben-Yehuda et al., 2012, Ferreira da Silva et al., 2013]. [Ferreira da Silva et al., 2013] propose a self-healing mechanism to handle operational incidents trough task replication. It specifically studies the long-tail effect issue, and proposes a new algorithm to control task replication. The proposed healing process is parametrized on real application traces acquired in production on EGI. Experimental results obtained in production conditions show that the proposed method speeds up execution up to a factor of 4.

2.3 Monte-Carlo simulation on grids

Previous work has been done on the parallelization of Monte-Carlo simulations on distributed architectures. Most approaches use static splitting which consists in the partitioning of the simulation into tasks of a given size that are assigned to computing resources. The partitioning can be done based on the total number of primary particles (events), as well as on the geometry of the simulation. Dynamic splitting usually consists in a initial distribution of tasks and some ulterior reassignment during runtime. A few examples of static and dynamic splitting for Monte-Carlo simulations are presented in the following.

2.3.1 Static splitting

[Maigne et al., 2004] present results obtained by running GATE in parallel on multiple processors of the DataGrid[14] project (EGI predecessor). This approach relies on static splitting and uses particle parallelism, i.e. the geometry information is replicated on each processor and particles are distributed equally between processors.

This static even distribution of particles may underexploit resources on heterogeneous platforms. To deal with this issue, [Rosenthal, 1999] 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. This technique requires that the total computing time, the number of available parallel nodes and their computing performance are known in advance so that T can be computed accordingly. On grid architectures, this knowledge is difficult

14. http://eu-datagrid.web.cern.ch

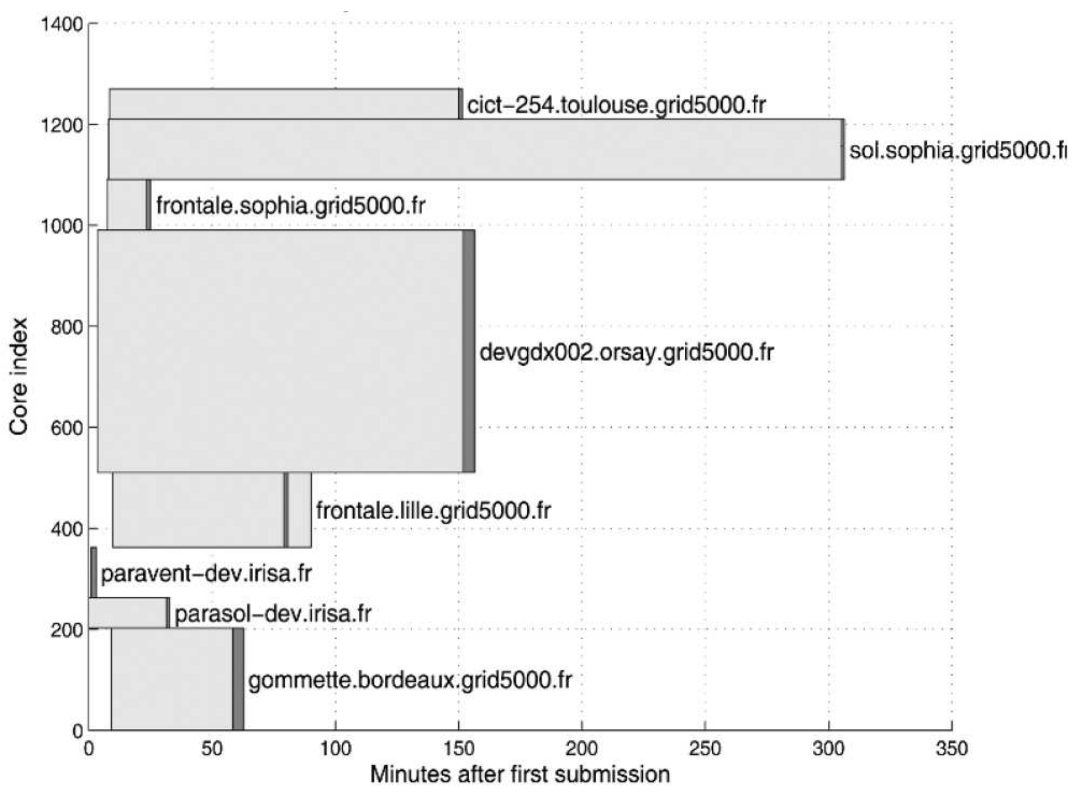


FIGURE 2.6: Queuing and executing times forMonte-Carlo jobs submitted on Grid5000, illustrating the difficulty in coordinated cross-site distributed computing. Light Grey boxes indicate queuing time, dark Grey boxes indicate execution time. Figure extracted from [Stokes-Ress et al., 2009].

to achieve. Moreover, the variable queuing times experienced on distributed architectures and presented in [Stokes-Ress et al., 2009] are definitely not compatible with this technique. Figure 2.6 illustrates the difficulties encountered with coordinated cross-site distributed computing, where tasks can be subject to significantly different waiting times and resource performance.

Static splitting can lead to very poor performance, in particular when a task is allocated to a slow resource towards the end of the simulation [Cirne et al., 2007]. This last task issue is worsened if failures occur and resubmission must be taken into account. Indeed, failures are recurrent on large grid infrastructures like EGI, where the success rate within the biomed VO has been noticed to be of the order of 80 to 85% [Jacq et al., 2008]. When splitting one Monte-Carlo simulation into sub-tasks, it is important that all of them complete successfully in order to retrieve the final result. Therefore, failed tasks must be resubmitted, further slowing down the application completion.

A possible solution to this problem is task replication [Cirne et al., 2007]. One replication method for grid-based Monte-Carlo calculations is presented in [Mascagni and Li, 2003b]. It uses the "N out of M strategy", i.e. it increases the number of subtasks from N to M.

Thus M parallel computations are submitted to the distributed environment. As soon as N results are ready, the final result can be produced. For Monte-Carlo applications, tasks are not literally replicated, since each task has a unique seed different from all others. In this sense, this solution could be seen as “over-computation”. Whatever the name, its main drawback is that it considerably increases the computational workload on the grid. Moreover, a good choice of M is not trivial since it varies from one application to another and it depends on the grid characteristics.

2.3.2 Dynamic splitting

Dynamic splitting tries to solve the load imbalance introduced by the static splitting on heterogeneous infrastructures by reassigning tasks or events to available processors during runtime.

In [Procassini et al., 2005], authors introduce a dynamic load balancing strategy for spatial parallelism. Spatial parallelism involves splitting the geometry into domains and then assigning a specific domain to one processor. This method is usually needed when the problem geometry has a significant size so that one processor does not have enough memory to store all particles/zones. Spatial parallelism may introduce load imbalance between processors, as spatial domains will require different amounts of computational work. The dynamic load balancing algorithm proposed in [Procassini et al., 2005] assigns multiple processors to each domain and then redistributes the number of processors among domains according to the evolution of the workload per domain. In this approach, communications are generated between processors in order to transmit changes from the last state. This implementation is therefore cluster oriented and is not adapted for grid usage where communications between processors are very costly.

[Galyuk et al., 2002] propose a dynamic approach using semaphores and an MPI implementation executed on multiple clusters under distributed memory conditions. Processors are guided by some control information concerning the number of events still needed to reach the initially required number of events. This information is rendered accessible through files that are transferred between the clusters. Given the communication and synchronization requirements of this approach, it is clearly not adapted for loosely-coupled grid usage, but could probably fit HPC systems.

[Bezzine et al., 2006] propose a dynamic approach which consists in splitting each computation into a large set of elementary tasks, that are dynamically distributed on the workers: each worker receives an elementary task to process and asks for a new one when it has finished. This is a classical pilot job approach, as described also in [Mościcki et al., 2011].

2.3.3 Science Gateways

Distributed systems are complex and difficult to use by end users who are not computer scientists. Science Gateways enable the exploitation of such complex systems by offering to end users a simplified interface: functionalities such as user authentication and access control, data transfer from local host to distributed storage, application parametrization, launching,monitoring and debugging become accessible with a few mouse clicks.

Among the existing life-science gateways, we can cite WeNMR[15] for bio-informatics applications, Neugrid [Frisoni et al., 2011] for medical imaging and related simulations, and P-Grade [Kacsuk, 2011] and GISELA/EUMEDGrid [Barbera et al., 2011] for more general purpose applications. Science Gateways for MonteCarlo applications usually target specialized communities, such as radiotherapists.

The E-IMRT platform described in [Josï¿½ Carlos Mouriï¿½o Gallego, 2007, Pena et al., 2009] offers radiotherapists a set of algorithms to optimize and validate radiotherapy treatments. It has three main components, namely characterization of linear accelerators, radiotherapy treatment planning optimization and verification and data repository. The IMRT treatment verification using Monte Carlo is a computational expensive problem that is an excellent candidate to be executed on a grid. These services are accessible through an user-friendly web page, where the implementation of the verification and optimization algorithms, as well as the complexity of the distributed infrastructure on which they run, are hidden to the user. The E-IMRT platform became one of the 25 Grid Business experiments (BEs) from the BEinGRID (Business Experiments in Grid) project. BEinEIMRT ([M.G. Bugeiro, 2009]) provides on-demand e-Health computational services (like tumor detection and radiotherapy planning) to Health organizations like clinics and hospitals. All services are provided by the Centro de Supercomputación de Galicia (CESGA), which is the customary provider. When CESGA is under peak demand, external resources are added for a limited time period[16]. This process is transparent for the end user, reducing execution time and increasing the QoS.

The HOPE (HOspital Platform for E-health) platform[17] was designed to enable GATE simulations in a grid environment. Its user-friendly interface was built taking into account feedback from healthcare professionals. Like the E-IMRT platform, it targeted especially physicians and medical physicists.

15. http://www.wenmr.eu/
16. http://www.ogf.org/documents/GFD.167.pdf
17. http://eu-acgt.org/news/newsletters/summer-2009/single-article/archive/2009/july/article/hope-hospital-platform-for-e-health.html

Researchers' requirements are different from those of the clinicians and medical physicists who are targeted by the E-IMRT and HOPE platforms. These platforms were mainly designed to perform the same (limited) set of (standard/validated) applications to several data (SIMD like approach), e.g. computing IMRT treatment plans on different patient data. In contrast, researchers provide macros files describing their simulations. They need to be able to easily test simulations with different set of parameters, to perform computing intensive simulations that lead to large phase-spaces or to study the design of new imaging devices, such the ones that will be developed to monitor the dose deposit in hadrontherapy situations thanks to the outgoing particles produced by nuclear interactions.

2.4 Evaluation methods

Evaluation methods can combine multiple approaches, such as experimentation in production or controlled environments, simulation and formal modeling. Experiments ensure realistic conditions and results, but are cumbersome and difficult to reproduce. Simulation becomes thus interesting, since it enables faster experimentation and controlled conditions. Modeling can help understand the behavior of distributed applications and tune parameters such as the checkpointing delay. Nevertheless, it remains a challenge for applications running in production conditions because most parameter values are unknown before the completion of the experiment.

2.4.1 Production experiments

Using a production infrastructure ensures that the realism of the experiments cannot be questioned. Nevertheless, to the best of our knowledge, experiments on large production infrastructures are rare and often restrained to illustrative results. Large production infrastructures are not always available for testing. If they are, production experiments can by prohibitively long and expensive, especially because they need a large number of repetitions in order to capture some of the variability of the experimental conditions, in particular the system load. Indeed, the lack of control over the conditions experienced on production infrastructures makes it difficult to reproduce the results, altering thus their significance.

[Chervenak et al., 2007] study the relationship between data placement services and workflow management systems. In order to demonstrate how data placement can improve the performance of workflows with large input data sets, authors compare the performance of a workflow that uses asynchronous data placement with one that does on-demand data placement. The experiments ran in production conditions, on a cluster with up to 50 available compute nodes.

In [Germain Renaud et al., 2008] authors propose and evaluate a virtualization scheme, as well as two user-level schedulers. The tests for the virtualization scheme are conducted on one EGI site (at LAL) to ensure the correct configuration. The tests for the user-level schedulers are split into approximately 40000 tasks performed simultaneously by around 200 worker agents at 6 EGI sites across Europe.

In the field of parallel and distributed computing, reproducing results is rarely seen as a fundamental step [Beaumont et al., 2011]. This is also the case in the previously presented papers where, despite the realism provided by the production conditions, results remain illustrative, since no repetitions are performed and since their reproducibility cannot be guaranteed.

2.4.2 Controlled environments

As presented in Section 2.2, there exists a certain number of experimental testbeds designed to support experiments in academia. These experimental grids support various grid and cloud software with and without virtualization and have the advantage of being user-customizable. Resources can be reserved in advance and users can have root access on the machines. In [Riteau et al., 2010], authors study sky computing based on Grid5000 and FutureGrid. Using the reconfiguration mechanisms provided by these testbeds, the Nimbus open source cloud toolkit can be deployed on hundreds of nodes in a few minutes. This gives access to cloud platforms similar to public infrastructures, such as Amazon EC2. Full control of the physical resources and of their software stack guarantees experiment repeatability. Many other experiments have used Grid5000, among which those presented in [Galis et al., 2011, Tang and Fedak, 2012, Samak et al., 2013, Vu and Huet, 2013].

Controlled environments are thus very interesting tools for the study and evaluation of performance achieved on distributed infrastructures. Nevertheless, one cannot guarantee that all assumptions are correct and that results obtained on such environments are reproducible on production infrastructures. Network status, resource availability and workload are only a few examples of parameters that can vary significantly between controlled environments and production conditions.

2.4.3 Simulation

There exists a variety of simulation toolkits providing APIs for the simulation of applications on distributed infrastructures, among which OptorSim [Bell et al., 2003], GridSim [Buyya and Murshed, 2002], PeerSim [Montresor and Jelasity, 2009], CloudSim [Calheiros et al., 2011], and SimGrid [Casanova et al., 2008]. Recent reviews of these tools are available, e.g., in [Beaumont et al., 2011] and in [Naicken et al., 2006] which focuses on peer-to-peer platforms. The related work presented in the following covers literature on (i) the platforms used in simulations, (ii) simulated grid services, (iii) application deployment and (iv) types of applications that have been simulated recently.

Platforms used in simulators are usually synthetized or modeled from existing ones [Lu and Dinda, 2003, Quinson et al., 2010], for instance using network tomography [Castro et al., 2004] to build a network model from a real platform. OptorSim has a model of the EU Datagrid platform (an ancestor of EGI), but it focuses on inter-site communications [Frincu et al., 2008]; computing sites are modeled as single processing units that can only process one job at a time [Mairi and Nicholson, 2006].

Computer scientists also use production logs as input for simulators. In [Mościcki et al., 2011], authors analyze the distribution of job waiting times in EGI and they perform simulations to compare their proposed late-binding (pilot-job) model with the classical job submission method. In [Ben-Yehuda et al., 2012], authors mention that to evaluate their Ex-PERT scheduling framework "in a variety of scenarios yet within our budget", they use simulated experiments coupled with data obtained from real-world experiments. Nevertheless, the fundamental question of how close the model is to the real world is not asked. Simulation can introduce a large bias, but very few authors have published extensive "simulation validation" results in the literature.

Various kinds of grid services were simulated. For instance, [Caron et al., 2007] report on a simulation involving a DIRAC pilot-job scheduler, and OptorSim and the work in [Sulistio et al., 2008] simulate data management services. A few MapReduce tools have been simulated too; for instance, [Wang et al., 2009] simulate Hadoop cluster and uses it to study the impact of data locality, network topology, and failures on applications. In [?], the authors propose a MapReduce simulator using GridSim.

The simulation of applications deployment, in particular scheduling, has received a lot of attention and custom tools have been developed, including GridSim [Buyya et al., 2005], and SimGrid [Santos-Neto et al., 2005] which was initially designed for the simulation of schedulers [Legrand et al., 2003].

Different types of application models have been simulated. SMPI [Clauss et al., 2011] is a SimGrid-based simulator for MPI applications in which the application is actually executed, but part of the execution is wrapped by the simulator. The work in [da Silva and Senger, 2011] simulates bag-of-task applications to study their scalability. In [Bouguerra et al., 2011], the authors use simulation to study the checkpointing of sequential applications to improve fault-tolerance.

2.4.4 Formal Modeling

Analytical models can also be interesting for performance evaluation.

In [Lingrand et al., 2010] authors try to improve grid application performances by tuning the job submission system. They propose a stochastic model, capturing the behavior of a complex grid workload management system instantiated using statistics extracted from grid activity traces. The traces used in this study have been collected by the Grid Observatory [18] and represent complete job runs with detailed information on the different states the job has encountered during its life cycle. Based on this information, the authors determine the ratio of outliers and the ratio of failed jobs, as well as the probability for a job to succeed. These values are then inserted into the model which computes the latency of successful jobs using resubmission in case of failures. The model is exploited for optimizing a simple job resubmission strategy.

[Lassnig et al., 2010] analyze the accuracy of existing prediction models on workloads and introduces an averages-based model to predict bursts in arbitrary workloads. The authors complement their own data sets with traces from the Grid Workload Archive [Iosup et al., 2008] (from which they select Grid5000 traces) and with synthetic data sets that demonstrate possible maximum and minimum boundaries. The proposed method consists in identifying and modeling the bursts. Once the candidates are identified, the shape of the burst is assessed using a modified gradient descent algorithm.

In [Glatard and Camarasu-Pop, 2009] authors present a performance model for pilot-job applications running on production grids. Given a number of submitted pilots and the distribution of the grid latency F_L , the model estimates (i) the evolution of the number of available pilots along time, i.e., the number of pilots that have reached a computing node and (ii) the makespan of the application. The model was evaluated on a Monte-Carlo application running on the EGEE grid (EGI predecessor) with the DIANE pilot-job framework [Mościcki, 2003]. The distribution of the grid latency F_L was measured from probe round-trip times.

18. http://www.grid-observatory.org

[Mościcki et al., 2011] analyze the makespan of applications executed on grids using classical grid early-binding and late-binding (pilot-jobs) systems. The authors propose theoretical models for the two systems, for which they derive the makespan L as a function of the queuing time τ, the total workload W and the processing power p, which is considered equal for all worker nodes. In addition to the model, simulations are also performed in order to compare the proposed late-binding model with the gLite meta-sheduler used in EGI

Models are sometimes implemented in simulation [Mościcki et al., 2011], but rarely confronted to production experiments [Glatard and Camarasu-Pop, 2009].

2.5 Conclusion

Monte-Carlo simulations are easily parallelized using a split and merge pattern, but existing splitting approaches are not well adapted to grids, where resources are heterogeneous and unreliable. In order to cope with this issues, we propose a parallelization approach based on pilots jobs and on a new dynamic partitioning algorithm. This new approach, together with results obtained on EGI, is presented in Chapter 3.

Related works on Monte-Carlo applications mostly focus on the computing part of the simulation while the merging of partial simulation results is also a critical step. On world-wide computing infrastructures, where partial results are usually geographically distributed close to their production site, the merging time can even become comparable to the simulation makespan. Chapter 4 presents advanced merging strategies with multiple parallel mergers. Checkpointing is used to enable incremental results merging from partial results and to improve reliability. The proposed strategies are evaluated in production conditions. A model aiming at explaining measures made in production is also introduced.

To the best of our knowledge, no simulation of applications using computing and storage resources on the production system of the European Grid Infrastructure has been performed. In Chapter 5, we describe our attempt to do so using SimGrid. Based on the extensive but cumbersome experience collected on the real system, we simulate the hardware platform, the core software services, the deployment, and the application of the previously described framework.

Last but not least, Chapter 6 illustrates the outcome of the proposed framework by giving some usage statistics and a few examples of results obtained in production.

Chapter 3

A Dynamic Load-Balancing Approach Using Pilot Jobs

Abstract *Particle-tracking Monte-Carlo applications are easily parallelizable on computing grids. However, advanced scheduling strategies and parallelization methods are required to cope with failures and resource heterogeneity. This chapter*[1] *presents a parallelization approach based on pilots jobs and on a new dynamic partitioning algorithm. Results obtained on EGI using the GATE application show that pilot jobs bring strong improvement w.r.t. regular metascheduling and that the proposed dynamic partitioning algorithm further reduces execution time significantly, i.e. by a factor of two in our experiments.*

3.1 Introduction

As presented in Chapter 2, static splitting approaches provide poor scheduling in heterogeneous non-reliable environments such as EGI. To illustrate this point, Figure 3.1 plots the task flow of a GATE simulation statically split into 75 tasks and executed on EGI. Load balancing is clearly sub-optimal: during the last hour of the simulation, at most 6 tasks run in parallel, which obviously underexploits the available resources. Heterogeneity has a dramatic impact leading to very long tasks (e.g. task 16) and very short ones (e.g. task 8). Moreover, errors lead to resubmissions that further penalize the execution.

We are thus seeking a dynamic load balancing strategy that would allow to distribute the computing charge on the available resources depending on their performance. Given

1. The results described in this chapter have been published in [Camarasu-Pop et al., 2010]: S. Camarasu-Pop, T. Glatard, J. T. Moscicki, H. Benoit-Cattin, and D. Sarrut, Dynamic partitioning of GATE Monte-Carlo simulations on EGEE, Journal of Grid Computing, vol. 8, no. 2, pp. 241-259, mar, 2010

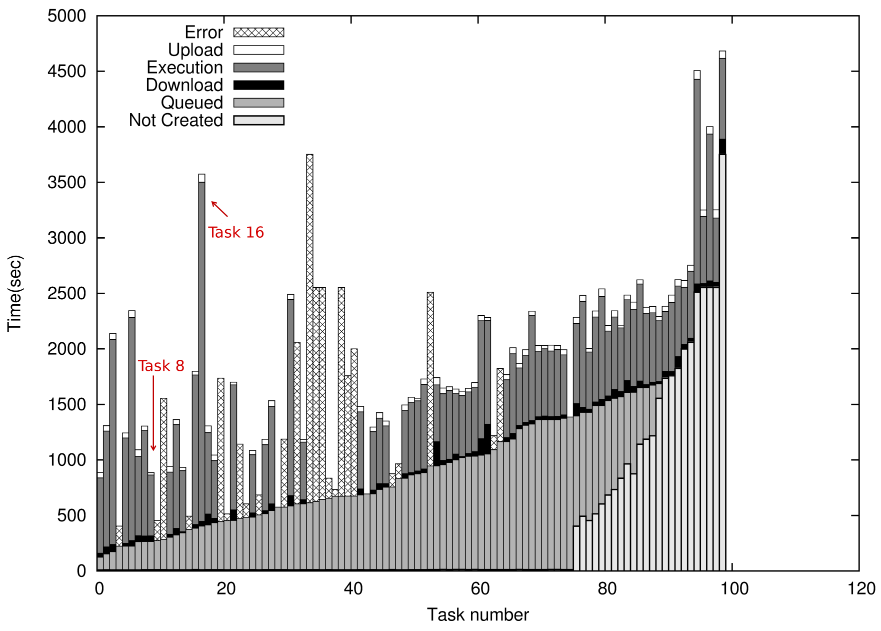


FIGURE 3.1: Example of task flow obtained with static partitioning of a GATE simulation executed on EGI. The simulation was split in 75 tasks but 24 of them failed for various reasons (4 data transfer issues, 6 pilots killed and 14 application errors). Hatched bars figure the time during which a failed task ran before the error happened. Tasks 76 to 99 are resubmissions, highly penalizing the performance. During the last hour of the simulation (from time 3000s to 4600s) at most 6 tasks run in parallel, which obviously underexploits the available resources.

the wide scale of the target grid infrastructure, communications among tasks, between tasks and the master and between tasks and output storage elements have to be avoided as much as possible. Moreover, task replication should be used sparingly on shared infrastructures such as EGI to limit the overhead on other users.

The rest of the chapter is organized as follows: next section presents a pilot-job strategy and its comparison to regular metasheduling. Our new dynamic load-balancing algorithm is introduced in Section 3.3, where we present its implementation using pilot jobs and the gain obtained when using our dynamic parallelization w.r.t a static approach. The chapter closes on a discussion and conclusions.

3.2 Pilot-job strategy

3.2.1 Method

A classical static splitting approach for particle-tracking Monte-Carlo simulations consists in dispatching the total number of events N into T tasks, each task receiving a fraction N/T from the total number of events. When using pilot jobs, the master assigns tasks to available pilots until all tasks are successfully completed. At the end of each task, pilots upload their result and are assigned a new task if available.

Pilot jobs can thus dynamically balance the number of events if they are assigned small, multiple tasks one after the other. A very fine task granularity (e.g. one event per task) is equivalent to a dynamic partitioning (each pilot would fetch and execute 1-event simulations until the whole simulation completes) but introduces a high overhead (communication with the master, in/output file transfer, application start-up, etc). Conversely, a coarse task granularity (large number of events per task, therefore small number of tasks compared to available resources) reduces the overhead but becomes equivalent to a static approach, leading to poor scheduling in heterogeneous non-reliable environments such as EGI (see Figure 3.1).

When using a pilot-job framework, the master automatically reassigns failed tasks to running pilots. The errors can be thus handled faster than with a standard metasheduling system where failed tasks are queued and have to wait for available resources.

Next section will present experiments and results when using static splitting with standard metascheduling and pilot-jobs.

3.2.2 Experiments and results

The experiments presented in this chapter aim at comparing performance achieved with DIANE pilot jobs w.r.t. gLite metascheduling (WMS). Experiments consist in executing a 16-hour[2] GATE simulation of 450,000 events. On-the-fly download, installation and execution of GATE on the worker nodes is performed. The executable and necessary shared libraries are packed into one tarball stored on one of the grid Storage Elements (SE) or on the master server and downloaded as soon as the job starts its execution.

In the following, two scenarios are compared. The first scenario (gLite) implements static parallelization with standard WMS submission. GATE tasks containing the same number of events are created and submitted to gLite as independent grid jobs (using the `glite-wms-job-submit` command). The second scenario (DS) implements the same static parallelization approach, but uses DIANE pilot jobs to execute the Gate tasks.

In order to evaluate the impact of the task granularity, we split the initial GATE simulation of 450,000 events into 25, 50 and respectively 75 tasks (i.e. 18000 , 9000 and respectively 6000 events per task). Each of the 2 scenarios consists thus of 3 different experiments and each experiment is repeated 3 times in order to capture some of the

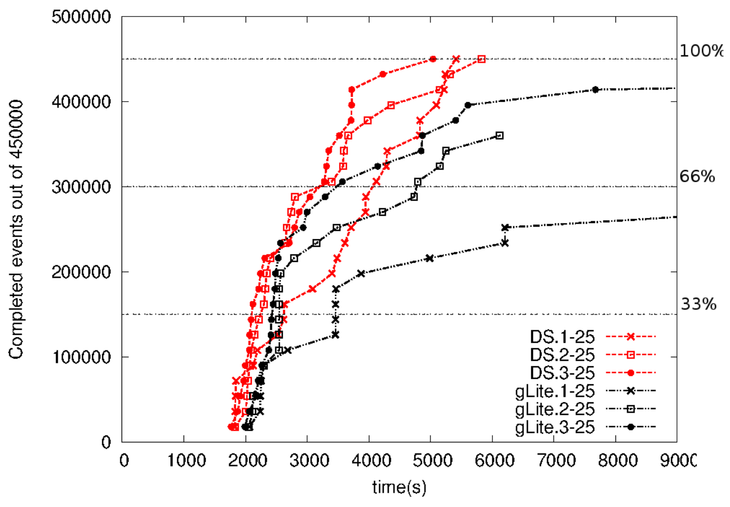
2. Average duration when executed on EGI resources

grid variability. Moreover, in order to compare scenarios in similar conditions, the corresponding repetitions of the 2 scenarios are submitted simultaneously. For comparison purposes, the number of submitted pilots (respectively gLite jobs) for each experiment corresponds to the number of tasks created for that experiment.

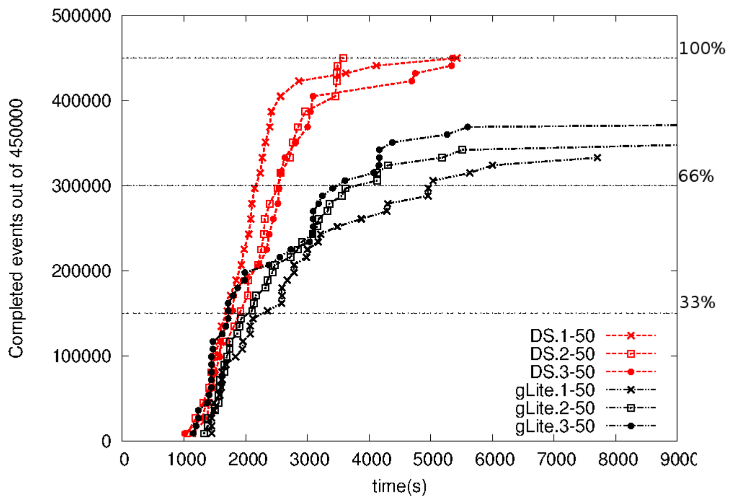
Failed jobs (DIANE pilots or gLite jobs) are not resubmitted. However, for the pilot-job implementation (second scenario), failed tasks are reassigned to registered pilots. Faulty pilots (i.e. pilots running a task that fails) are removed from the master pool to avoid new failures and are not resubmitted. In realistic conditions, job resubmission is essential for the classical metascheduling approach and can be highly desirable for the static pilot-job scenario in order to replace the faulty pilots. In our case we chose not to do it for comparison purposes so that the same number of jobs are submitted to the infrastructure in both cases.

Figure 3.2 plots the application completion rate, i.e. the number of computed events along time for the two scenarios and with different numbers of submitted jobs (i.e. different task granularity). It shows that the two scenarios are roughly equivalent for the first third of the simulation regardless of the number of pilots. At the beginning of the simulation, the two scenarios benefit equally from the fastest EGI resources. From 33% to 66%, gLite begins to worsen and after 66% it degrades drastically both in terms of performance and in terms of reliability. This is due to the fact that the pilot jobs continue to exploit resources by reassigning new tasks to them, wheras the gLite scenario releases resources as soon as they finish their unique task. Scenario 1 (gLite) never manages to complete 100% of the simulation. Indeed, the success rate is in most of the cases between 70% and 80%. These results are confirmed for all three cases (25, 50 and 75 jobs or pilots). As expected, pilot jobs bring dramatic improvement w.r.t. default metasheduling, both in terms of reliability and performance.

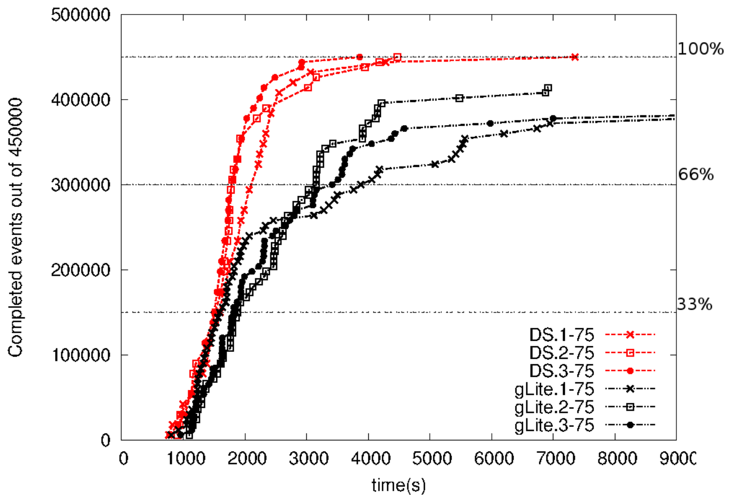
For the pilot-job (DS) scenario, Figure 3.2(b, c) shows that the computing throughput (events/s) begins to significantly slow down around 400,000 events (90%). This is due to late resubmission of failed tasks and to the assignment of tasks to slow resources towards the end of the simulation. This corresponds to the issue of the last tasks, as presented in Chapter 2 Section 2.3, and it will be addressed by the dynamic partitioning method proposed in the next section.



(a) 25 pilots



(b) 50 pilots



(c) 75 pilots

FIGURE 3.2: DIANE implementation vs regular gLite-WMS implementation. The two scenarios are drawn with a different line style and experiments conducted in parallel are plotted with the same symbols (star, circle or square). On each graph three horizontal lines are drawn at 33%, 66% and 100% of the running jobs and of the simulated events respectively. gLite performance degrades drastically after 66% and never manages to reach 100% of the simulation. The naming convention corresponds to scenarioName.repetitionNumber-numberOfJobs.

Algorithm 1 Master algorithm for dynamic load-balancing of Monte-Carlo simulations

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

3.3 Dynamic load-balancing algorithm

3.3.1 Algorithm

The proposed dynamic-load balancing consists in a ”do-while” loop with no initial splitting. Each independent task of a simulation is created with the total number of events and keeps running until the desired number of events is reached with the contribution of all the tasks. Therefore, each task may simulate the whole simulation if the other tasks fail or do not start. The total number of simulated events is given by the sum of all events simulated by the independent tasks. Thus, each computing resource contributes to the whole simulation until it completes.

Tasks have all the same inputs but a different random seed number which allows each task to simulate a unique set of events complying with the simulation properties. Thus, since different tasks of the same simulation are statistically independent, they generate independent sets of events that can be merged.

Algorithms 1 and 2 present the master and the pilots pseudo-code. The master periodically sums up the number of simulated events and sends stop signals to pilots when needed. Each pilot executes only a single task, starting as soon as the pilot reaches a worker node and stopping at the end of the simulation. Light communications between tasks and master occur periodically to upload the current number of events computed by the task and at the end of the simulation when the master sends stop signals.

Errors are efficiently handled by this algorithm. When a task fails, the remaining ones just keep running until all the events have been simulated. No task resubmission is thus required.

3.3.2 Implementation for the GATE application using pilot jobs

To implement our dynamic algorithm, the GATE code had to be extended to handle stop signals during simulation. This add-on allows the application to pause regularly (at

Algorithm 2 Pilot algorithm for dynamic load-balancing of Monte-Carlo simulations

```
Download input data
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
```

a period that can be specified in the configuration macro file and that corresponds to the `updateDelay` from Algorithm 2), launch an auxiliary program and wait for its exit code. Depending on this exit code, GATE resumes or stops the simulation. In our case, the auxiliary program checks if the stop signal has been received and, if this is the case, the GATE sub-simulation saves its results and stops.

The dynamic approach follows the algorithms presented in Algorithms 2 and 1. Each GATE task is initially assigned the total number of events N. The master has a counter for already simulated events that is updated every time the pilots upload their current status. The simulation is complete when the total number of events is reached.

Pilot jobs provide the necessary framework allowing the jobs to easily communicate with the Master. Nevertheless, they are not a pre-requisite for implementing our dynamic load-balancing algorithm. The algorithm could be deployed using metascheduling with a dedicated communication overlay.

3.3.3 Experiments and results

Experiments reported in this section aim at identifying the gain provided by our dynamic load balancing algorithm. To address this question, we compare the classic static parallelization (scenario 2 from previous section) with our dynamic parallelization, both approaches using DIANE pilot jobs. We will refer to the static scenario as DS and to the dynamic one as DD.

As for Section 3.2.2, each experiment consists in executing a 16-hour[3] GATE simulation of 450,000 events. Both approaches are tested with 25, 50 and respectively 75 pilot jobs and an equal number of tasks (i.e. 25, 50 and respectively 75 tasks). For the static

3. Average duration when executed on EGI resources

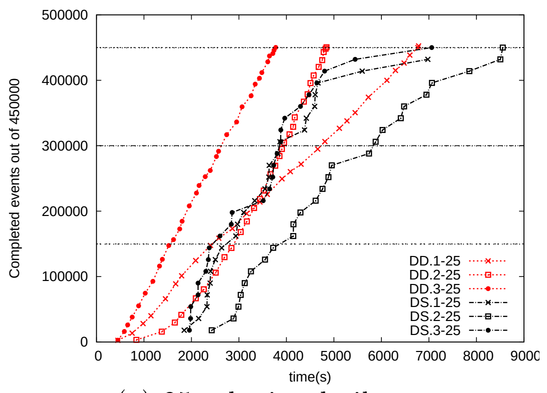
approach, this implies that simulations are split into tasks computing 18000, 9000 and respectively 6000 events. For the dynamic approach, all tasks are assigned the total of 450,000 events. Each of the 2 scenarios consists of 3 different experiments repeated 3 times and submitted simultaneously similarly to the experiments in Section 3.2.2.

Faulty pilots (i.e. pilots running a task that fails) are removed from the master pool to avoid new failures and are not resubmitted. For the static implementation, the failed tasks are reassigned to registered pilots. For the dynamic implementation, task resubmission is not necessary since pilots that do not fail compute until the end of the simulation and since we submit a number of pilots equal to the number of tasks: resubmitted tasks would remain queued until the end of the simulation.

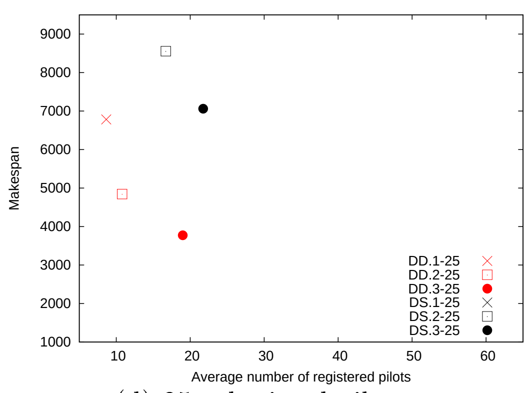
Figure 3.3 compares the performance of the dynamic and static implementations. We can see that the dynamic parallelization brings significant performance improvement, the makespan being on average 2 times smaller for the experiments with 75 pilots. Indeed a significant amount of time is saved on the completion of the last tasks. Thanks to the lack of resubmission and better resource exploitation, the computing throughput (events/s) stays constant until the end of the simulation for the dynamic approach. In Figure 3.3 (a), the poor performance of the first repetition of the dynamic scenario with 25 pilots (DD.1-25) is due to the small number of registered pilots.

The number of registered pilots influences the performance and is dependent on the grid conditions. It could be thus argued that, due to the variability of grid conditions, the difference in performance observed in Figure 3.3 comes from the differences in grid conditions. To check that this is not the case and that better performance is actually achieved due to our dynamic load-balancing algorithm, we measured the number of registered pilots for each experiment.

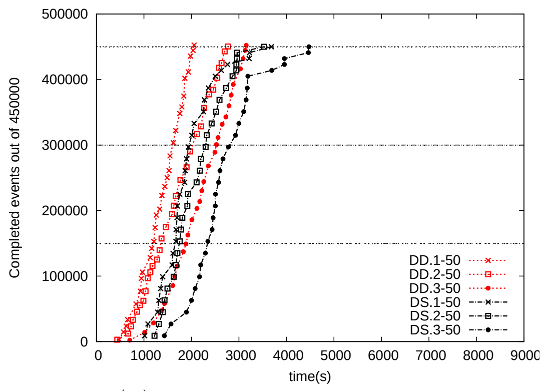
Figure 3.4 displays the number of registered DIANE pilots along time. Overall, it shows that no scenario was favored by the grid scheduler although there is some variability among repetitions. In Figure 3.4(a) most of the runs have similar behavior, with about half of the submitted pilots registered in less than 15 minutes (900 seconds). We notice however two singular runs: the second repetitions of both scenarios with 25 pilots (DS.2-25 and DD.2-25). They correspond to 2 different scenarios executed simultaneously at a moment when the grid must have been more loaded than for the other runs. This illustrates the fact that experiments run simultaneously are subject to similar grid conditions. The first repetition of the dynamic scenario with 25 pilots (DD.1-25) is clearly an outlier with very few registered pilots, which penalizes the overall performance of the experiment as discussed previously (see comments on Figure 3.3). In Figure 3.4(b), DS.3-50 and DD.3-50 are two other singular runs corresponding to simultaneously executed experiments for which pilots register with similar throughput, but a longer latency. In



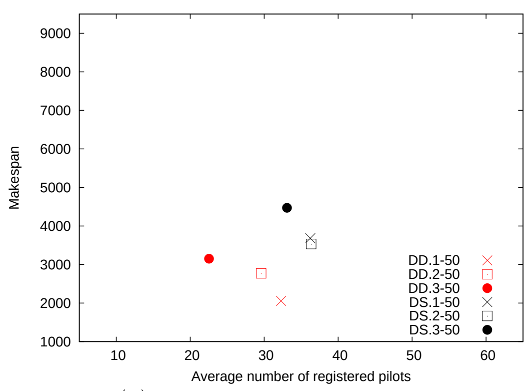
(a) 25 submitted pilots



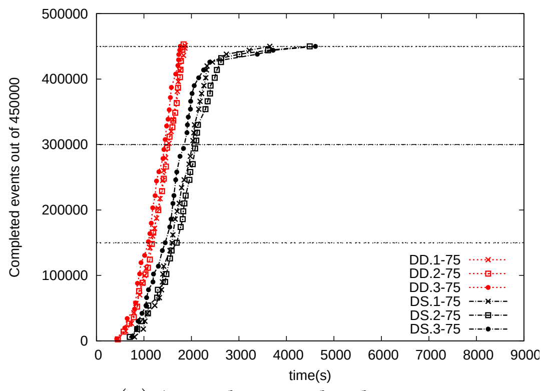
(d) 25 submitted pilots



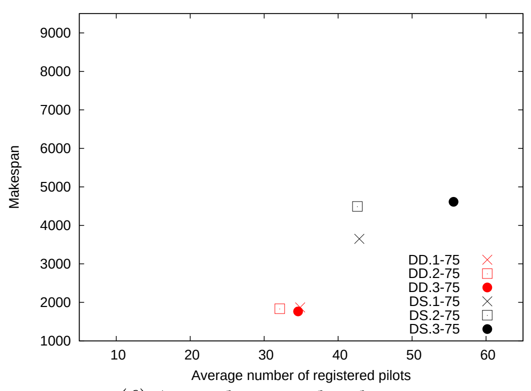
(b) 50 submitted pilots



(e) 50 submitted pilots

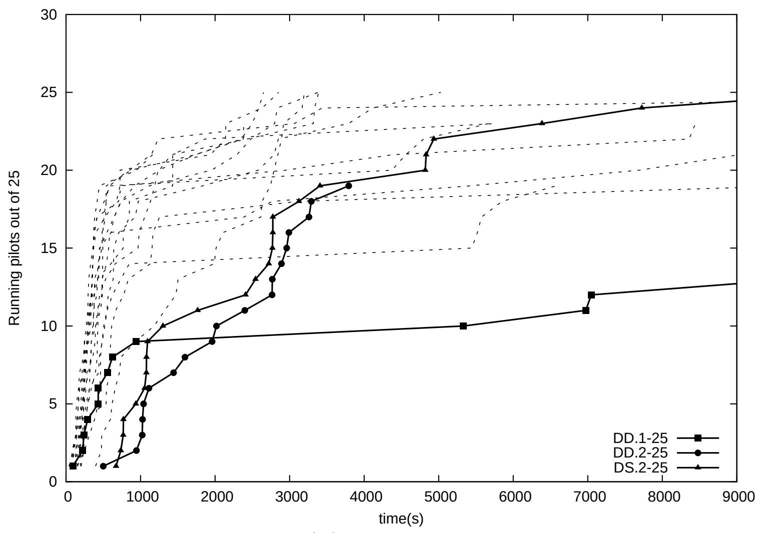


(c) 75 submitted pilots
Completed events along time

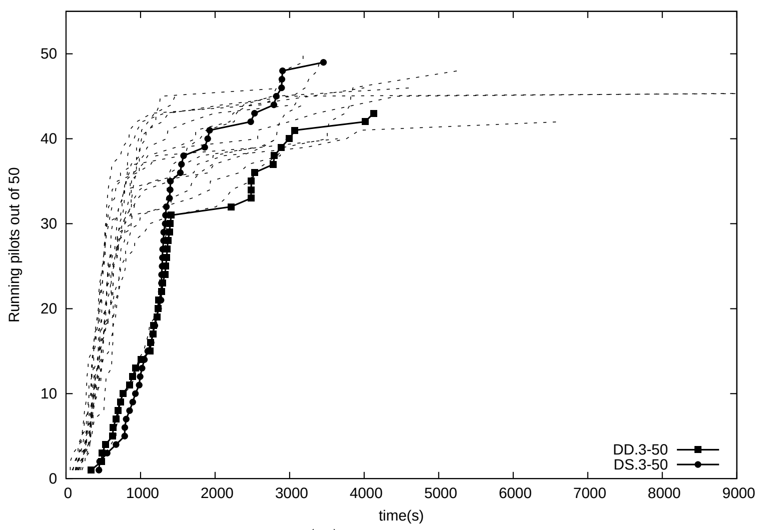


(f) 75 submitted pilots
Makespan w.r.t registered pilots

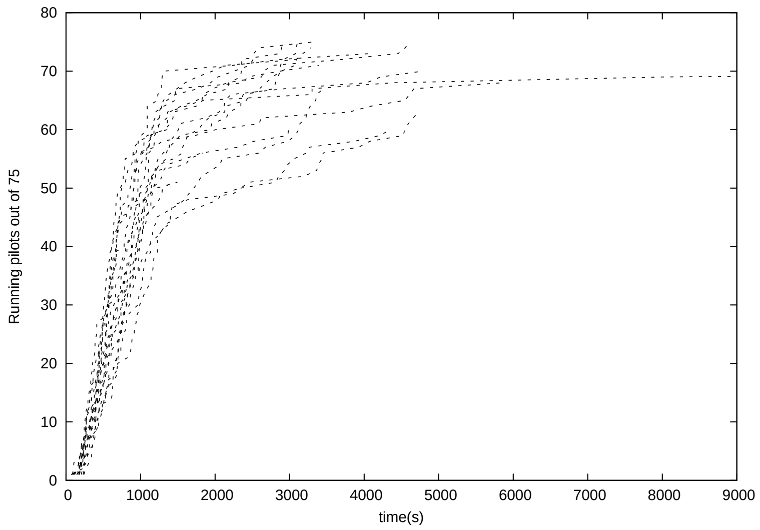
FIGURE 3.3: DIANE dynamic implementation vs DIANE static implementation. The two scenarios are drawn with a different line style and experiments conducted in parallel are plotted with the same symbols (star, circle or square). On each graph three horizontal lines are drawn at 33%, 66% and 100% of the running jobs and of the simulated events respectively. The dynamic parallelization brings significant performance improvement as the makespan is considerably (up to two times) smaller. The naming convention corresponds to scenarioName.repetitionNumber-numberOfPilots.



(a) 25 pilots



(b) 50 pilots



(c) 75 pilots

FIGURE 3.4: Registered pilots along time. Dashed lines plot runs for which the evolution of the number of registered pilots is similar. Dark lines plot runs showing singular behaviors (outliers). Overall no scenario is favored by the grid scheduler.

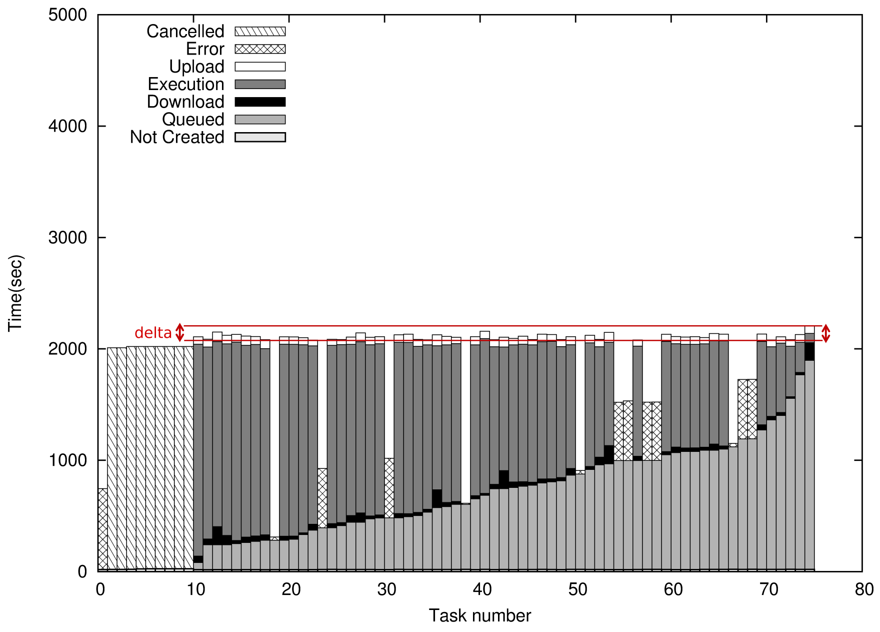


FIGURE 3.5: Example of task flow obtained with dynamic partitioning of a GATE simulation on EGI with 75 submitted pilots. Available resources are all exploited until the end of the simulation. Errors are compensated without task resubmissions. The dynamic load balancing obviously outperforms the one obtained with static partitioning (Figure 3.1).

Figure 3.4(c) the evolution of the number of registered jobs is overall very homogeneous among runs.

We can thus conclude that for these experiments the difference in performance actually comes from the splitting method and not from the differences in grid conditions.

In Figure 3.3-(d,e,f) we plotted the makespan [4] as a function of the average number of registered pilots during the simulation. We notice that for the same splitting method, performance is better (makespan is smaller) when more pilots register. This is less obvious for the static approach, where the simulation makespan highly depends on the slower tasks and failures, being thus very sensitive to grid conditions and less predictable.

Figure 3.5 plots a task flow obtained with dynamic partitioning. The scheduling obviously outperforms the one obtained with static partitioning (see Figure 3.1). Tasks complete almost simultaneously and errors are compensated without resubmissions. The dynamic load balancing succeeds in compensating the problems (errors, spare or slow resources) of the static approach.

4. In Figure 3.3(d) due to an experimental problem (temporary connectivity issues with the master) we were unable to determine the makespan of the first repetition of the static scenario with 25 pilots (DS.1-25)

3.4 Discussion

When using the proposed dynamic load-balancing algorithm, computing resources are fully exploited until the end of the simulation: resources are released simultaneously up to a time interval `delta` defined as the time difference between the first and the last task completion; `delta` is bounded by the longest propagation time of the `stop` signal and the longest upload time. When `delta` can be neglected, our load-balancing algorithm can be considered optimal both in terms of resource usage and makespan. In Figure 3.5, `delta` is inferior to 3 minutes, representing 6% of the simulation makespan.

The proposed algorithm may lead to computing slightly more events than initially needed, since each pilot computes more than one event during the `updateDelay`. It may also happen that, upon receiving the stop signal, a pilot cannot upload its results. In this case, its contribution is lost and the final result may contain slightly less events than initially needed. According to Monte-Carlo application experts, computing slightly less or more events is not a problem as long as the exact number of events that have actually been computed is known so that the result can be interpreted accordingly.

On heterogeneous distributed infrastructures, where resources cannot be reserved in advance, pilots will experience different conditions (waiting and running times, performance, errors) for different executions of the same application. In these conditions, and since the stop condition is based on the number of events contributed by all pilots, the number of events computed by a given pilot will vary from one execution to another. This renders impossible the exact reproduction of a Monte-Carlo simulation executed with the dynamic load-balancing algorithm, even if random seeds are fixed for each pilot.

The proposed algorithm can only be used for Monte-Carlo applications for which all events are strictly equivalent and can be simulated in any order. Nevertheless, there also exist Monte-Carlo applications for which events are not strictly equivalent, usually because of time or spatial dependencies. Some of these applications could be adapted to our algorithm by randomly selecting the events to simulate from the complete set of events. For instance, the Monte-Carlo simulation of water diffusion in the cardiac fibers for DMRI [Wang et al., 2012b] consisted initially in simulating, for a number of V voxels, N events uniformly distributed in each voxel. In this configuration, events are not strictly equivalent, since they belong to a given voxel. If we tried to execute it with the proposed dynamic approach, all pilots would start simulating events in the same voxels and would be stopped when the sum of events computed by all pilots would reach the required value of NxV, i.e. before the completion of the V voxels. In order to render it compatible to the proposed dynamic load balancing algorithm and to execute it in production [Wang

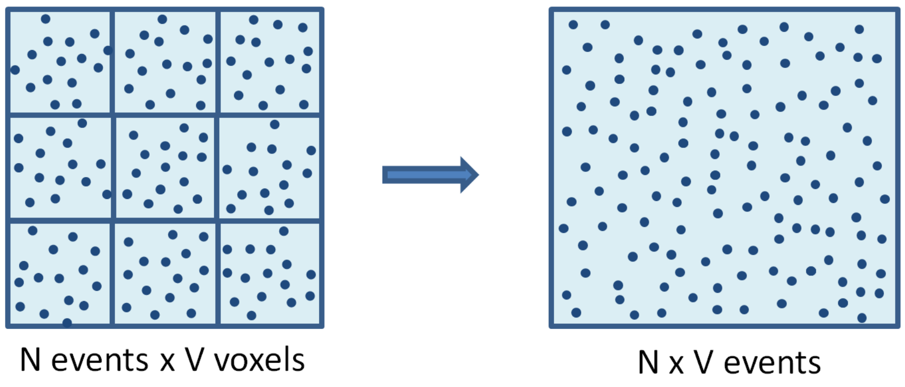


Figure 3.6: Example of a possible adaptation to render Monte-Carlo applications compatible with the dynamic load balancing algorithm. Initially, the application computes N events uniformly distributed in each of the V voxels which are treated sequentially. Thus, if all pilots begin simulating events in the same voxel and are stopped before the completion of the V voxels, no event will be computed in the last voxel(s), even if the sum of events computed by all pilots would be greater than the total required NxV events. To solve this issue, the voxels can be considered as a unique volume inside which NxV uniformly distributed events are computed.

et al., 2012a], the application has been slightly modified, so that it simulates NxV events uniformly distributed over the V voxels as described in Figure 3.6.

3.5 Conclusion

This chapter presented a dynamic load-balancing approach for particle-tracking Monte-Carlo applications. Although these applications are easily parallelizable, the heterogeneity of distributed infrastructures leads to recurrent errors and high latencies. To address these problems, we proposed a new dynamic load-balancing method and its implementation using pilot jobs for its integration on EGI. The algorithm consists in a ”do-while” loop with no initial splitting. The master periodically sums up the number of simulated events and sends stop signals to pilots when the total number of requested events is reached. Thus, all tasks complete almost simultaneously, rendering our method optimal both in terms of resource usage and makespan. Results show that the proposed algorithm brings speed-ups in the order of two w.r.t conventional static splitting.

Consequently, we consider that the proposed dynamic approach solves the load-balancing problem of particle-tracking Monte-Carlo applications executed in parallel on distributed heterogeneous systems, up to a communication time interval `delta` and the limitations discussed at the end of section 3.3.1. Nevertheless, the computing phase has to be followed by a merging phase allowing to retrieve the final results. Next chapter will present the issue of the merging phase and will propose new solutions for further improving the overall performance of parallel Monte-Carlo applications.

Chapter 4

Parallel and Incremental Merging with Checkpointing Strategies

Abstract *Chapter 2 proposed a dynamic load balancing approach that significantly reduces the execution makespan by efficiently using available resources until the end of the simulation. However, this algorithm only considers the Monte-Carlo phase, which is followed by a merging phase consisting in downloading and merging all partial results into one final output. Depending on the number of partial results, their availability and transfer times, the merging phase can significantly increase the application makespan.*

The present chapter[1] *enriches the framework proposed previously by integrating advanced merging strategies with multiple parallel mergers. Checkpointing is used to enable incremental results merging from partial results and to improve reliability. The performance of the proposed strategies are evaluated with respect to the complete makespan, i.e. including both the computing and merging phases. A model is proposed to analyze the behavior of the complete framework and help tune its parameters. Experimental results obtained on EGI using GATE 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. The results described in this chapter have been published in [Camarasu-Pop et al., 2013b]: S. Camarasu-Pop, T. Glatard, R. Ferreira da Silva, P. Gueth, D. Sarrut, and H. Benoit-Cattin, Monte-Carlo Simulation on Heterogeneous Distributed Systems: a Computing Framework with Parallel Merging and Checkpointing Strategies, Future Generation Computer Systems, vol. 29, no. 3, pp. 728–738, 03/2013

4.1 Introduction

The merging phase of a Monte-Carlo simulation parallelized on a grid consists in downloading and merging all partial results into one final output. Since partial results are usually geographically distributed close to their production site, their transfer time is very sensitive to the availability of storage resources, and to the network throughput and latency [Ma et al., , Li et al., 2010]. The merging problem is exacerbated by the quasi simultaneous production of partial results at the end of the computing phase, since the dynamic load balancing uses all computing resources until the end of the simulation.

To illustrate that, Figure 4.1 shows the job flow of a GATE simulation. Computing jobs finish almost simultaneously, approximately at the same moment when the merging job starts its execution. Here the merging time represents approximately 20% of the total makespan, but in some other cases, due to extremely long transfer times of some of the partial results, it can even become comparable to the simulation makespan. Our goal is thus to deliver the merged result as soon as possible after the end of the Monte-Carlo phase.

As mentioned in Chapter 2 Section 2.2, checkpointing is often used to improve application reliability. In our case, checkpointing is also interesting for result merging because it enables incremental merging of partial results during the computing phase. Nevertheless, checkpointing has to be properly tuned in order to limit the overheads. A too short checkpointing period would over-charge the mergers due to the very large number of checkpointed results, while a too long checkpointing period would reduce the benefit of incremental merging.

In the following, we propose and evaluate two methods to deal with the merging problems: (i) multiple parallel mergers and (ii) checkpointing with incremental merge. We integrate these strategies in the framework initiated in Chapter 3 and we test them on EGI in order to ensure realistic assumptions and results. For a deeper analysis, we also introduce a model describing the behavior of the different scenarios and allowing to analyze their performance beyond the experiments made in production.

The rest of the chapter is organized as follows: the complete framework integrating parallel, incremental merge with simulation checkpointing and dynamic load-balancing is described in Section 4.2. In Section 4.3 we propose a model aiming at explaining the performance of the framework (measures) in production. In Section 4.4, experiments evaluate the impact of merging strategies and checkpointing in production conditions. The chapter closes on a discussion and conclusions.

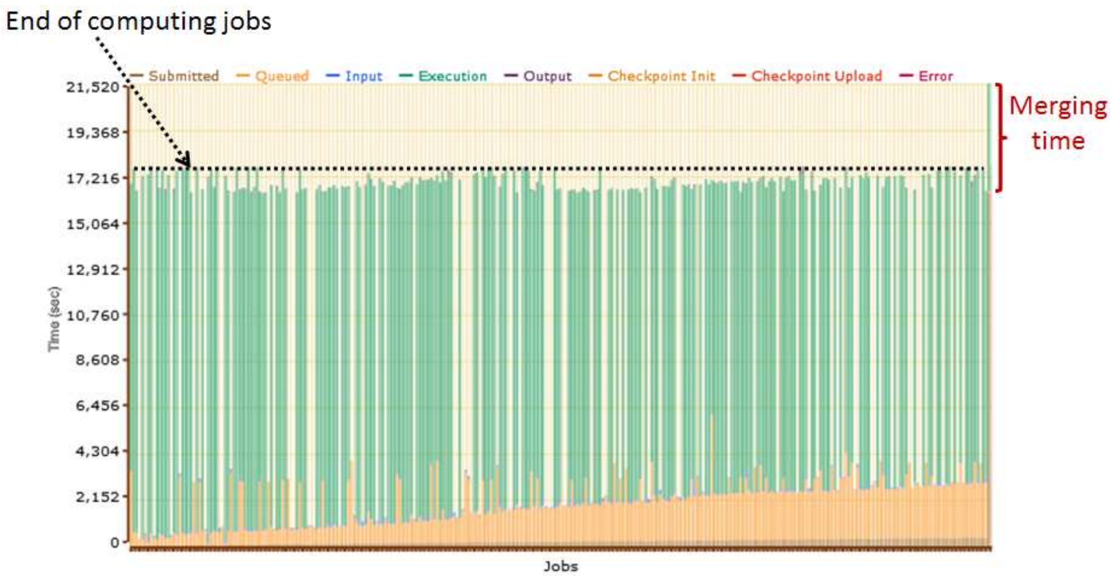


FIGURE 4.1: Job flow for a GATE simulation without checkpointing and with one merging job. Computing jobs finish almost simultaneously, approximately at the same moment the merging job starts its execution. Empty bars correspond to jobs for which we lack information on the running and completion times (it may happen for some error, stalled or canceled jobs).

4.2 Merging Strategies

4.2.1 Multiple parallel mergers

In the Monte-Carlo applications considered here, the merging process is commutative and associative. We can therefore use multiple parallel mergers in order to reduce its makespan. This strategy distributes not only the merging operation itself, but also the transfers of the partial results.

Figure 4.2 presents the framework introduced in Chapter 3 and extended with multiple mergers. The master generates multiple parallel tasks containing the total number N of events to simulate. Each task periodically transmits its current number of simulated events to the master. At this point, these events reside on the local disk and are not available for merging. Simulation tasks then check the stopping condition given by the master when the total number of simulated events from all simulation tasks is reached. This part of the framework corresponds to the dynamic load balancing algorithm presented in Chapter 3. When the stopping condition is reached, the master launches multiple merging tasks and the simulation tasks upload their partial results into a shared logical folder (LFC). Note that the content of this folder may be physically stored on multiple, distributed storage elements. While there is no `StopMerge` signal, the merging tasks select, download and merge batches of partial results. Their result is then uploaded back

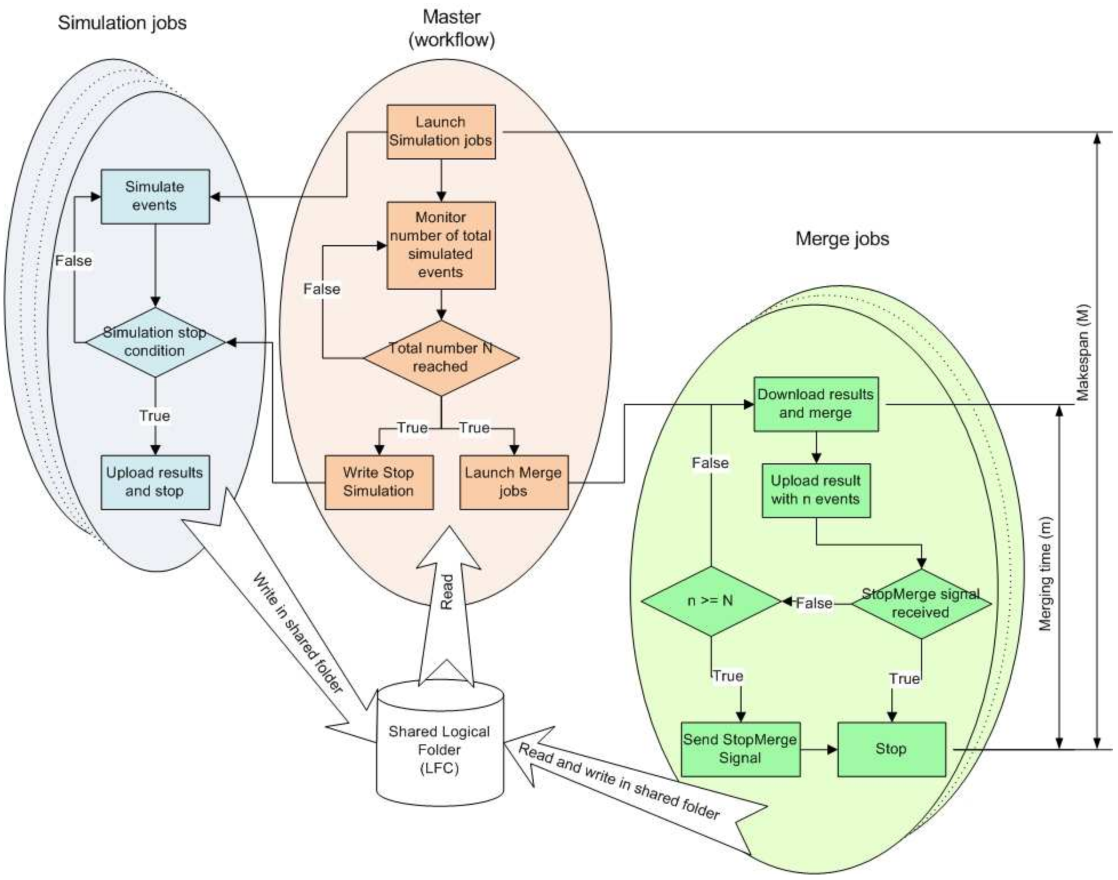


FIGURE 4.2: Framework without checkpointing. Simulation tasks are represented in blue, merging tasks in green and the master in orange. Multiple simulation and merge tasks run in parallel but, for readability reasons, only one simulation and one merge tasks are represented. Simulation tasks upload their results once at the end of their lifetime. The SimulationStop condition given by the master triggers: i) results upload from simulation tasks and ii) the launch of merging tasks. The StopMerge signal is sent by the first merging task having merged the required number of events.

to the shared folder for subsequent merging. If a merge task produces a result containing the total number of events, then it also sends the `StopMerge` signal.

4.2.2 Checkpointing

In order to enable incremental merging of partial results, we propose to use checkpointing. The framework enriched with checkpointing is described in Figure 4.3. Note the following important differences with the version without checkpointing:

– Simulation tasks checkpoint and upload their partial results at the checkpoint period.
– The master monitors the number of checkpointed events instead of simulated events (see Table 4.1 for the difference between simulated and checkpointed events).

Type of events	Description
Simulated events	Only reside on the local disk of a simulation task and are not available for merging.
Checkpointed events	Are uploaded to the shared logical folder and are therefore available for merging.
Merged events	Have been processed at least once by a merger. They continue to be merged incrementally until the final result, containing the total number of events, is produced.

TABLE 4.1: Different types of events involved in the framework.

– Merging tasks are launched from the beginning of the workflow because partial results are available since the first checkpoint. Apart from that, merging tasks are the same as without checkpointing.

In both configurations, each parallel merging task 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 framework and it is the same for all merging tasks. For a good load balancing of the merging activity, nf should be smaller than the total number of files to merge divided by the total number of mergers. Indeed, load is better distributed in smaller chunks as explained in Chapter 2. Nevertheless, the smaller the chunks, the higher the overhead. In our case, if files are merged two by two (nf=2), many intermediate results will be generated, increasing the number of file transfers and, consequently, the total transfer time. In the following, nf will be fixed experimentally.

Since the merging process is commutative and associative, 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 tasks and the partial results produced by computing tasks 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.

4.3 Model

A model of the makespan of Monte-Carlo executions integrating the merging strategies introduced previously is presented here. It aims at explaining measures made in production.

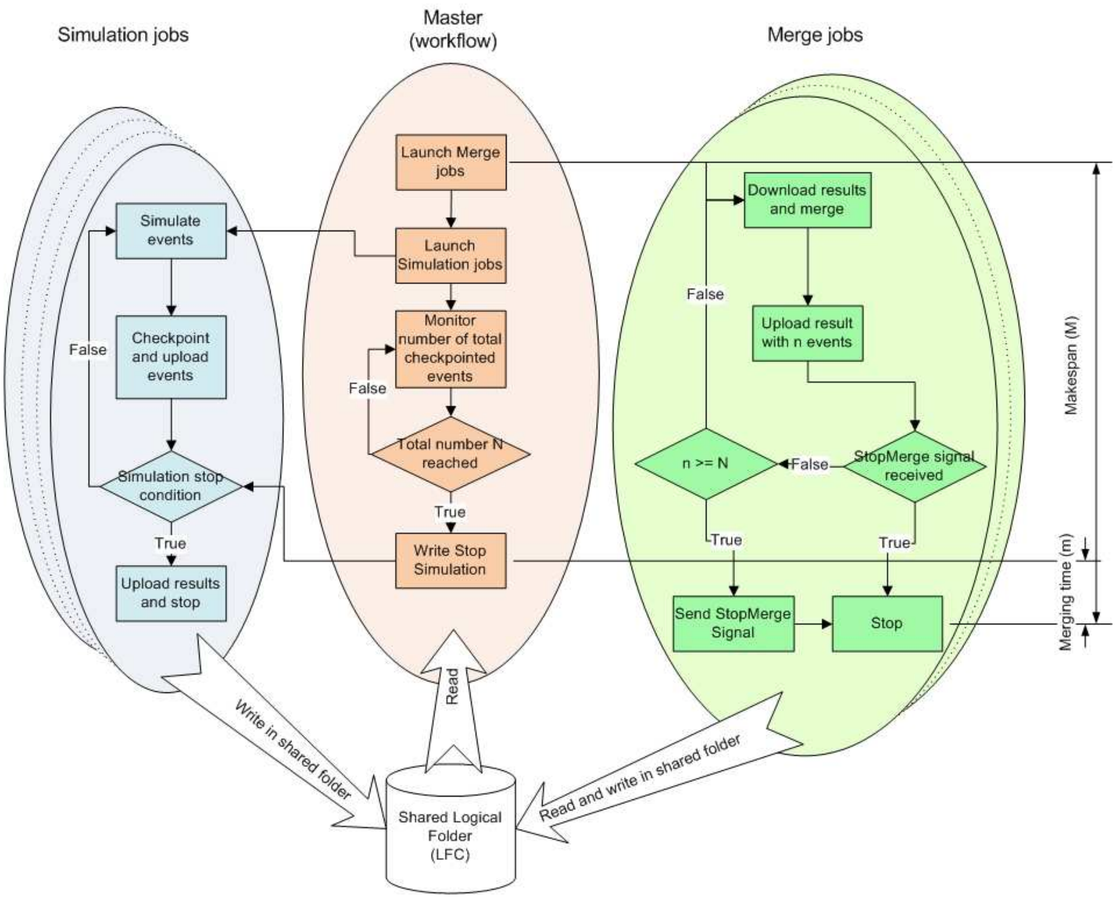


Figure 4.3: Framework with checkpointing. Simulation tasks are represented in blue, merging tasks in green and the master in orange. Multiple simulation and merge tasks are launched in parallel at the beginning of the workflow. For readability reasons, only one simulation and one merge tasks are represented. Simulation tasks 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 tasks. From this moment on, only merging tasks continue to run during the "extra merging time". The `StopMerge` signal is sent by the first merging task having merged the required number of events.

4.3.1 Model with multiple parallel mergers

The workload Γ of a Monte-Carlo simulation is freely divisible and, according to Chapter 3, the dynamic load-balancing algorithm behaves as if 1-event tasks were continuously distributed to n parallel jobs. In this case, the simulation tasks finish quasi simultaneously, as it can be seen on the job flow in Figure 4.1. In these conditions, if runtime errors, grid submission failures, and latency variability are ignored, the makespan M can be modeled as the average waiting time plus the average running time of the n parallel jobs : $M = \frac{\Gamma}{n} + E_L$ as explained in [Mościcki et al., 2011].

If we take into account that with a failure rate ρ, only $n(1-\rho)$ tasks 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, we have:

$$M = \frac{\Gamma}{n(1-\rho)} + E_L + m \tag{4.1}$$

In the following, the value of the merging time m will be measured from real experiments.

4.3.2 Model with checkpointing

If checkpointing is activated, then failed tasks 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 jobs time to failure (TTF) 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 checkpointing period of a simulation task. We assume that c is fixed and cannot be changed during the simulation. Since simulation tasks do not start simultaneously due to their individual queuing times, their checkpoints are not synchronized either. The end of the simulation is given by the last checkpoint of the task which contributes enough to reach the simulation stop condition, while the other tasks are still computing.

Knowing that all computing tasks are submitted at the beginning of the workflow and that there is no resubmission of failed tasks, there exists at least one task which checkpoints its results throughout the simulation. Let k be the number of checkpoints made by this task. k is the integer such that $kc \leq M - m - E_L < (k+1)c$, i.e.

$$k = \lfloor \frac{M - m - E_L}{c} \rfloor \tag{4.2}$$

The total CPU time Γ consumed by the simulation is the sum of the expected CPU times checkpointed by simulation tasks:

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

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

$1 - F(kc)$. Thus:

$$\Gamma = nc\left(k - \sum_{i \leq k} F(ic)\right) \tag{4.3}$$

Note that Γ only consists of the simulation time until the last checkpoint ($t = kc$). The time between the last checkpoint ($t = kc$) and the end of the simulation is ignored.

According to the experimental data that will be presented in Section 4.4.3 (see Figure 4.8), 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 Equation 4.3:

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

Note that this expression is easily extended to the case where F is linear only from t=pc (p <k). Using Equation 4.2, the makespan can be expressed using the following approximation:

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

So that:

$$M = \frac{\Gamma}{n\left(1 - \frac{F(kc)+F(c)}{2}\right)} + \frac{c}{2} + m + E_L \tag{4.4}$$

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

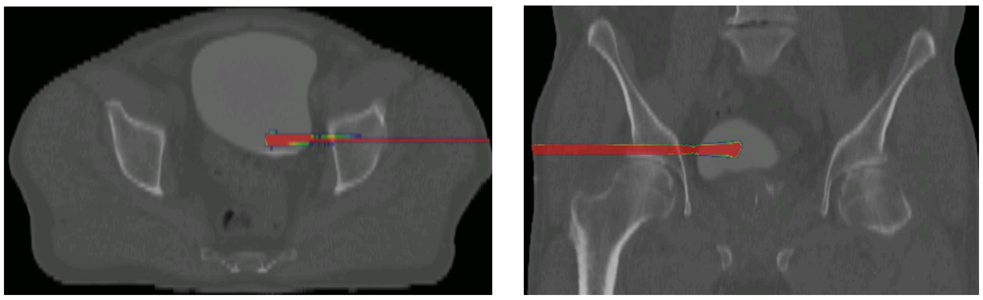

(a) Patient CT, axial view (b) Patient CT, coronal view

FIGURE 4.4: 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.

4.4 Experiments and results

4.4.1 Implementation

Both scenarios, with and without checkpointing, were implemented using MOTEUR and both workflows are integrated into the VIP/GATE-Lab web portal described in Chapter 6. Tasks are executed using DIRAC pilot jobs on the resources available to the biomed VO. 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.

4.4.2 Experimental conditions

Experiments were conducted with a proton therapy simulation using GATE. Figure 4.4 shows the result produced by a 50M events simulation. It represents the dose distribution obtained by a proton pencil beam as described in [Grevillot et al., 2011].

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. When executed with 300 parallel tasks, the task duration is superior to 24 hours. We measured the number of simulated and merged events (see Table 4.1) with and without checkpointing.

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

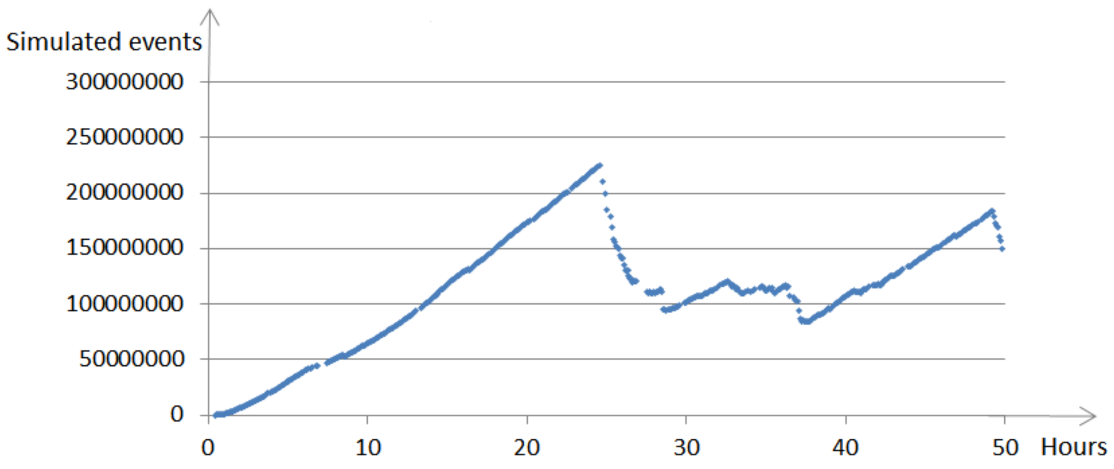
For the last two experiments, we measured the total CPU time of the simulation (Γ), the mean job waiting time (E_L), the merging time (m), the failure rate of simulation jobs (ρ) and the fraction of simulation jobs that failed before the first checkpoint ($F(c)$) for the simulations with checkpointing.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

Experiments were conducted on the production platform described in Chapter 6. Experiments were repeated three times to capture some of the grid variability.

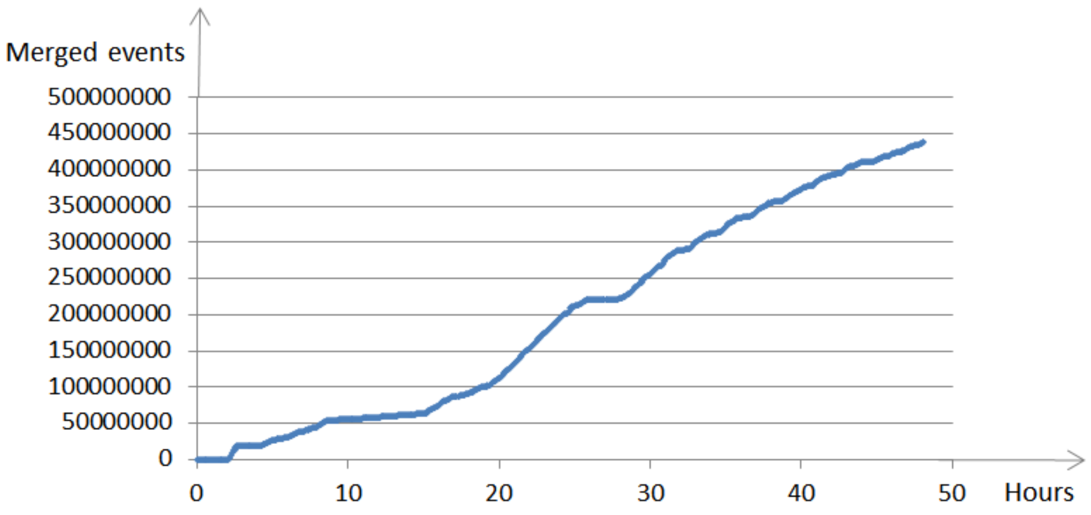
4.4.3 Results

4.4.3.1 Added value of checkpointing (Exp 1)

Figure 4.5 shows the number of merged and simulated events along time. As explained in Table 4.1, 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 framework without checkpointing (Figure 4.5-a) the number of simulated results increases steadily during the first 24 hours. 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 hours. In this experiment, the framework without checkpointing is simply not able to complete the simulation of 440M events.



(a) Framework without checkpointing



(b) Framework with checkpointing

FIGURE 4.5: Results for the experiment with a very long simulation (Exp 1), representing roughly one year of CPU time. The framework without checkpointing (a) is not able to complete the simulation because events from killed jobs are entirely lost. For the framework with checkpointing (b), the number of checkpointed events increases steadily till the end of the simulation.

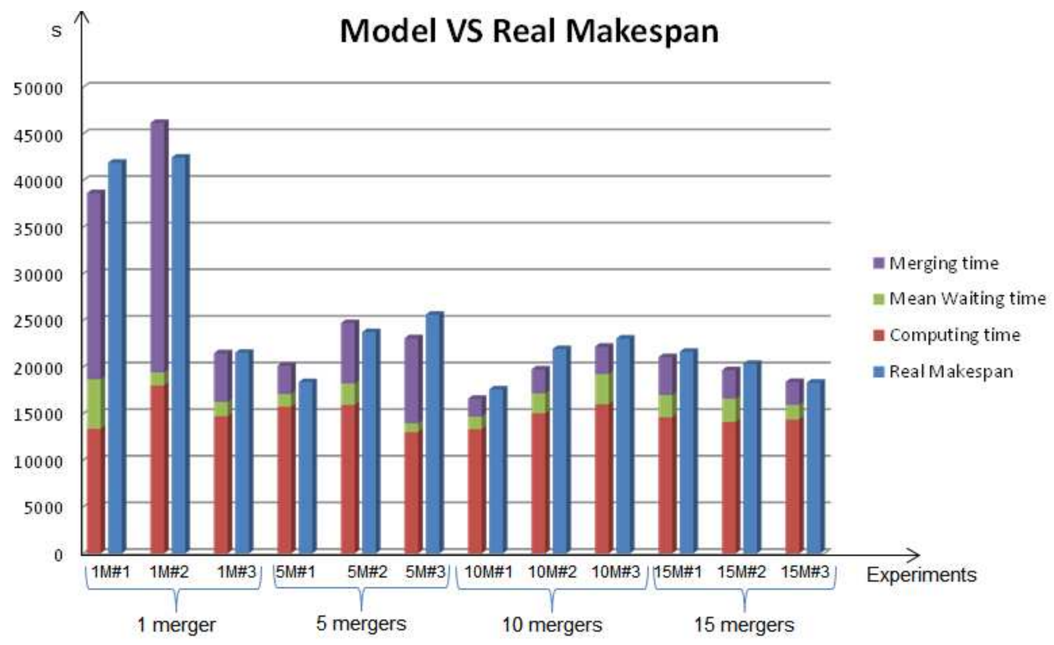


FIGURE 4.6: Results for the experiment without checkpointing (Exp 2). For each of the twelve GATE simulations (three repetitions for each of the four runs) two bars are printed: the blue bar on the right represents the real (measured) makespan in seconds, while the stacked bar on the left represents the makespan computed using the proposed model for GATE without checkpointing. The three stacked elements correspond to the three terms in Equation 4.1.

Conversely, the framework with checkpointing (Figure 4.5-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.

4.4.3.2 Impact of multiple mergers (Exp 2)

Table 4.2 details all measures obtained from Exp 2, and Figure 4.6 compares the measured makespan with the makespan computed from the model in Equation 4.1. The model percent error is computed as $(Mreal - Mmodel)/Mreal * 100$. 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 experimental data could not be properly fitted.

The results show that using a unique merger is clearly sub-optimal. From Table 4.2 we can see that the average makespan with one merger can be reduced by 40% when using 10 parallel mergers (from 35301 to 20861 seconds). 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.

run	ρ	E_L (s)	m (s)	M real (s)	M model (s)	Model error (%)
1 merger #1	0.148	5298	19980	41916	38653	7.8
1 merger #2	0.317	1404	26760	42460	46166	-8.7
1 merger #3	0.180	1554	5220	21528	21490	0.2
1 merger mean	-	-	17320	35301	35436	-
5 mergers #1	0.187	1361	3060	18384	20159	-9.7
5 mergers #2	0.191	2295	6480	23742	24718	-4.1
5 mergers #3	0.140	950	9120	25613	23092	9.8
5 mergers mean	-	-	6220	22579	22656	-
10 mergers #1	0.102	1346	1920	17601	16588	5.8
10 mergers #2	0.171	2143	2580	21927	19749	9.9
10 mergers #3	0.213	3240	2940	23055	22176	3.8
10 mergers mean	-	-	2480	20861	19504	-
15 mergers #1	0.128	2369	4080	21637	21061	2.7
15 mergers #2	0.123	2483	3060	20343	19659	3.4
15 mergers #3	0.150	1580	2460	18326	18406	-0.4
15 mergers mean	-	-	3200	20102	19709	-

Table 4.2: Experiment results – multiple parallel mergers (Exp 2).

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	21803	23968	-9.7
30 min #2	0.239	0.211	1026	7680	25214	27372	-9.1
30 min #3	0.202	0.184	3643	2640	23470	23794	-0.7
30 min mean	-	-	-	4460	23495	25024	-
60 min #1	0.183	0.177	1343	1080	20056	20744	-3.5
60 min #2	0.196	0.164	2501	840	24163	21724	10.6
60 min #3	0.206	0.193	1453	1860	22060	23584	-6.2
60 min mean	-	-	-	1260	22093	21972	-
120 min #1	0.120	0.100	4319	720	26071	27872	-6.4
120 min #2	0.263	0.241	4879	900	26368	25813	1.6
120 min #3	0.279	0.250	5336	900	28378	27241	3.9
120 min mean	-	-	-	840	26939	26990	-

Table 4.3: Experiment results – 10 parallel mergers with checkpointing (Exp 3).

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.

4.4.3.3 Influence of the checkpointing period (Exp 3)

Table 4.3 shows measures obtained from Exp 3, and model values computed using Equation 4.4, assuming $F(kc) = \rho$. Figure 4.7 compares the measured and modeled makespan. Overall the model correctly explains the experiments, with a relative error lower than 10%.

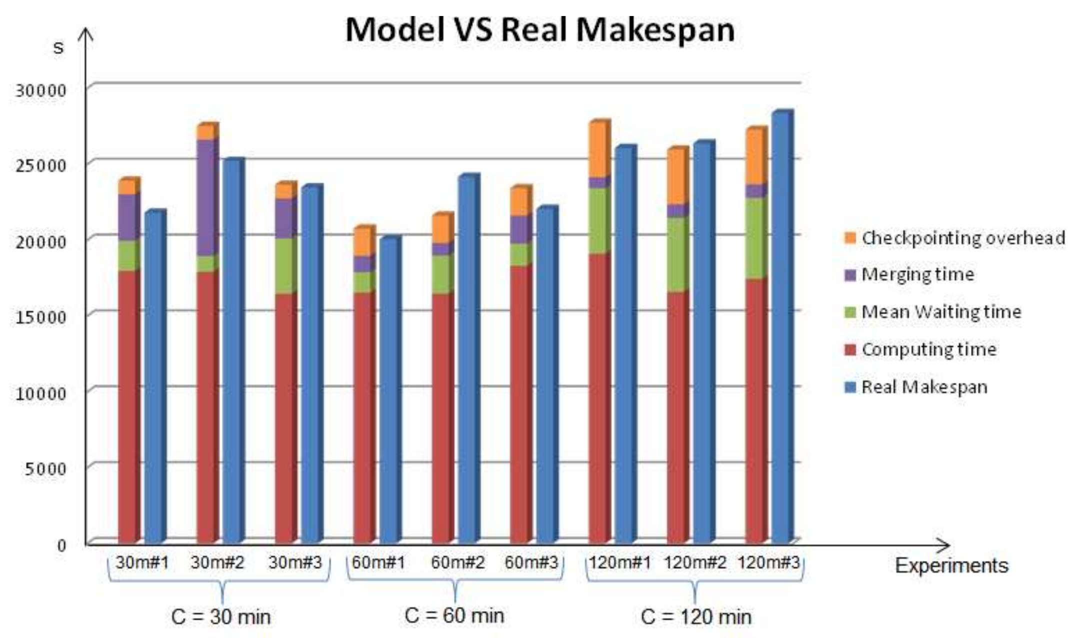


FIGURE 4.7: Results for the experiment with checkpointing (Exp 3). For each of the nine GATE simulations (three repetitions for each of the three runs) two bars are printed: the blue bar on the right represents the real (measured) makespan in seconds, while the stacked bar on the left represents the makespan computed using the proposed model for GATE with checkpointing. The four stacked elements correspond to the four terms in Equation 4.4.

From Table 4.3 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 checkpoints are too frequent. At the same time, according to the model and to Equation 4.4, 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 minutes provides the best average makespan.

Note that the merging time with checkpointing periods of 60 minutes is significantly smaller than without checkpointing and 10 mergers (in average, 1260s versus 2480s). Indeed, as illustrated on Figure 4.3, with checkpointing, the merging tasks run concurrently with the simulation tasks, 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 period as modeled by Equation 4.4 and observed on Figure 4.7. This overhead could be reduced if the checkpointing period was 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.

The influence of checkpointing is closely related to the failure distribution F, which is estimated as follows:

$$F(t) = \frac{\text{failed jobs of duration} < \text{t}}{\text{failed jobs of duration} < \text{t} + \text{jobs of duration} > \text{t}} \tag{4.5}$$

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 $\tilde{F}$:

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

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

Figure 4.8 plots the time-to-failure distribution F(t) for the 9 repetitions in this experiment. 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 tasks when they are waiting for new results checkpointed by simulation tasks.

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

4.5 Conclusion

This chapter integrated two merging strategies in the Monte-Carlo framework introduced previously. The merging step of a Monte-Carlo simulation can represent a significant amount of additional time in production conditions. To address this problem, we proposed an approach using multiple parallel mergers. Checkpointing was also proposed to enable incremental results merging from partial results and to improve reliability.

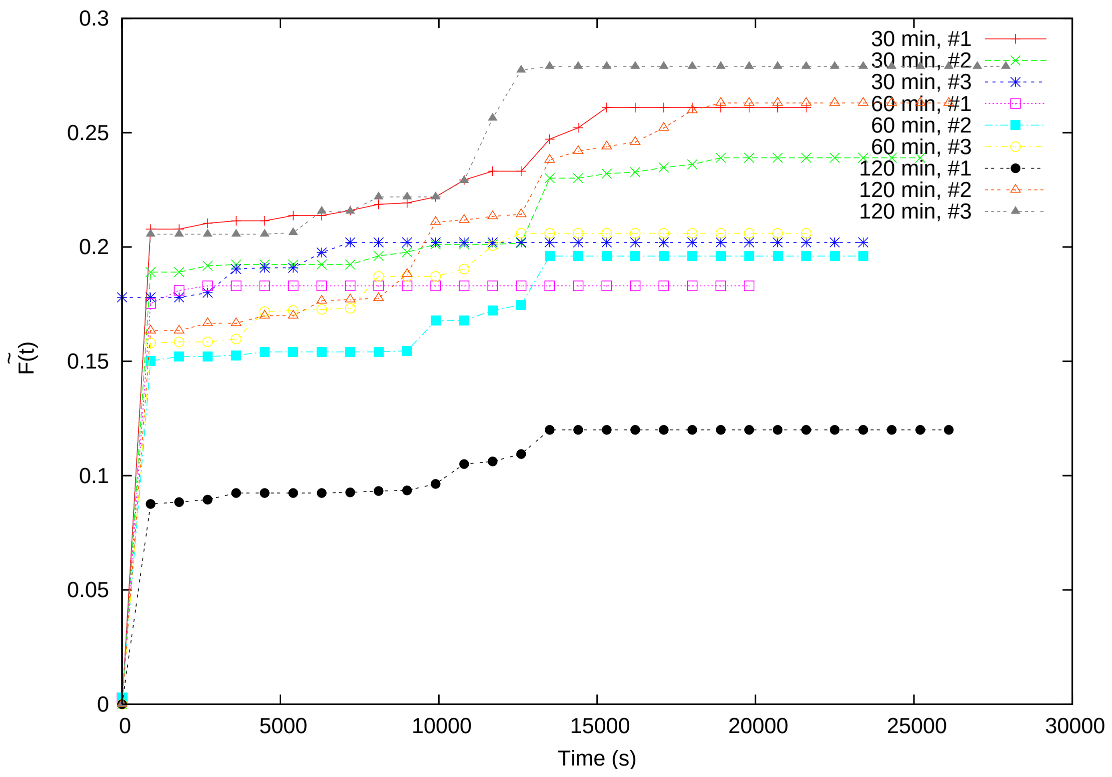


FIGURE 4.8: Measured TTF cumulative distribution on the 9 executions with checkpointing.

Three experiments have been conducted on EGI. 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 period, 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 period for the framework with checkpointing. The model is not 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%. Note that the merging time m is not modeled, but is integrated into the model with its real, mesured value. Its modeling would require knowledge on the number of mergers, their latency, error and file transfer rates, as well as on the number of files to merge, which is, at its turn, dependent on the frequency period.

All the results were obtained in production conditions, on the European Grid Infrastructure. In fact, the GATE framework with multiple mergers is already in production and executed by several users daily as will be discussed in chapter 6. While experimenting in production ensures that all the assumptions are realistic, it also limits reproducibility and statistical significance. To address this limitation, the next chapter will aim at reproducing these production experiments in a simulation environment based on the SimGrid toolkit. In order to ensure realistic simulations, we (i) implement our framework in the simulation environment, (ii) parametrize the simulation using traces captured from our real experiments (e.g. errors, job waiting times), and (iii) validate performance results obtained in simulation against real ones.

Chapter 5

Simulating Application Workflows and Services Deployed on EGI

Abstract *Chapter 4 proposed advanced merging strategies validated in realistic conditions on EGI. While experimenting in production ensures that assumptions are realistic, such experiments can be very long and cumbersome. Moreover, the variability of grid conditions renders experiment reproducibility very difficult. Simulation becomes thus interesting, since it enables faster experimentation and controlled conditions.*

The present chapter[1] *presents an end-to-end SimGrid-based simulation of the previously described framework for Monte-Carlo computations on EGI. Middleware services, namely the file catalog, storage elements, the DIRAC pilot-job system, and the MOTEUR workflow engine are simulated by SimGrid processes. The deployment of pilot jobs, performed on EGI by the gLite WMS and batch queues, is simulated by a random selection of platform hosts, and a matching of their latencies and failure times with real traces. The Monte-Carlo application workflow is calibrated to address performance discrepancies between the real and simulated network and CPUs. The simulation is evaluated against real executions of the GATE Monte-Carlo application. Results show that SimGrid can be used to study the performance of applications running on EGI. Simulated and real makespans are consistent, and conclusions drawn in production about the influence of application parameters such as the checkpointing frequency and the number of mergers are also made in simulation. These results open the door to better and faster experimentation.*

1. Results described in this chapter have been published in [Camarasu-Pop et al., 2013a]: S. Camarasu-Pop, T. Glatard, and H. Benoit-Cattin, S̈imulating Application Workflows and Services Deployed on the European Grid Infrastructure ̧ 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13), Delft, The Netherlands, 2013

5.1 Introduction

Scientific experimentation with large distributed platforms such as EGI, BOINC [2] or the Amazon Elastic Computing service [3] is a real challenge because they are subject to unpredictable load from user communities, to software and hardware failures of various types, and to dynamic resource availability. To address these difficulties, computer scientists rely on mathematical models [Mościcki et al., 2011, Glatard and Camarasu-Pop, 2009], simulators [Bell et al., 2003, Buyya and Murshed, 2002, Casanova et al., 2008], or experimental platforms [Bolze et al., 2006] to try to reproduce real systems in controlled conditions, which remains a challenge [Gustedt et al., 2009, Epema, 2012].

The aim of this chapter is to investigate how executions of application workflows deployed on EGI can be reproduced and studied using simulation. The ultimate goal of this study is to be able to replay real executions launched from the VIP/GateLab platform in order to understand, study and improve their performance. We do not aim at simulating the complete EGI system, but only the subset of platform and services used by a particular execution of an application that allows its reproduction and investigation in simulation.

Based on the extensive but cumbersome experience collected on the real system (see chapter 4), we build a simulator based on the SimGrid toolkit [Casanova et al., 2008] to simulate the hardware platform, the core software services, the deployment, and the application. The platform is the set of hardware resources used by the application: storage, network, and CPU. Services are software processes that are re-used by different applications, e.g., a workflow engine, a job scheduler or a file catalog. Deployment corresponds to the matching between software services and platform entities. Finally, the application encompasses all the processes specific to a computation performed by a user.

The platform is assumed already modeled, and we focus on the services, deployment and application. Differences between the real and simulated platforms, in particular CPU and network characteristics, are compensated by calibrating the simulated application on the simulated platform.

The rest of the chapter is organized as follows: Section 5.2 describes the real system under investigation. Section 5.3 presents the main functionalities of SimGrid. Section 5.4 details the design of our simulation and its calibration. Section 5.5 validates our simulator based on a comparison with real executions and Section 5.6 studies the influence of the checkpointing period based on more exhaustive results obtained with the simulator. The chapter closes on a discussion and conclusions.

2. `http://boinc.berkeley.edu`
3. `http://aws.amazon.com/ec2`

5.2 Real System to Simulate

We consider the real system made of the complete Monte-Carlo framework described in the previous chapters, deployed on EGI using the MOTEUR workflow engine and the DIRAC job scheduler.

5.2.1 Platform

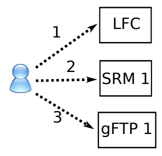
The target platform is the biomed Virtual Organization (VO) of EGI, introduced in Chapter 2. A detailed description of the biomed VO platform is provided by EGI[4]. On this platform, application jobs are often delayed by a few minutes to a few hours before being executed. We call this delay *latency*, which encompasses scheduling time, queuing time in batch systems and other overheads. Application jobs are also subject to failures that can occur at any time due to software or hardware issues.

5.2.2 Services

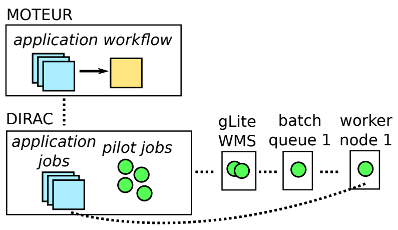
As summarized in Figure 5.1, the platform is accessed through a set of core services offering basic operations for job and storage management. Figure 5.1(a) illustrates file transfer services and their interactions. Storage resources are accessed through a three-layer stack. At the lowest level, files are transferred using the gridFTP transfer protocol. On top, the Storage Resource Manager (SRM) schedules file transfers on the gridFTP backends available on the site. Finally, a central logical file catalog (LFC) stores information about file replicas, and provides a single logical addressing space for distributed storage. File transfers are performed by gLite clients wrapping operations involving these 3 entities [Laure et al., 2006]. Numbers on Figure 5.1(a) indicate steps of a file download: (1) storage site is obtained from logical file name by querying the LFC; (2) transfer request is issued to SRM; (3) transfer is done through a gridFTP server. File upload follows a similar reverse sequence.

Figure 5.1(b) gives an overview of workload management services and their deployment. Jobs are scheduled on computing resources using the DIRAC [Tsaregorodtsev et al., 2009] pilot-job framework. Application jobs are generated, submitted and monitored by the MOTEUR workflow manager [Glatard et al., 2008b] from a description of the application workflow. MOTEUR jobs transfer the input data, perform various checks on the computing resource, run the application code itself, and finally upload the results on the site storage elements. MOTEUR also measures parameters about the execution,

4. `http://gstat.egi.eu/gstat/gstat/vo/biomed`



(a) File transfer



(b) Workload management.

FIGURE 5.1: Services and deployment on the real system

such as the latency and CPU power of each job, and timestamps of various execution phases (see details in [Ferreira da Silva and Glatard, 2012]). Failed jobs are automatically resubmitted up to a configurable number of times.

5.2.3 Deployment

Deployment of storage services is done statically by the resources providers. Typically, each site hosts a storage element made of one SRM server backed by one or several gridFTP servers. A single central LFC is available in the biomed VO.

The MOTEUR workflow engine and DIRAC scheduler are also statically deployed on hosts of the same network. Pilot jobs, however, are dynamically deployed through the resource-provisioning mechanism implemented in DIRAC: they are submitted either directly to queues of the sites' batch schedulers or through the gLite Workload Management System (WMS) which dispatches them on sites. DIRAC also plays a role in this dispatching, by using site blacklists defined by platform administrators. The pilot deployment process results from a combination of meta and local scheduling processes configured by various admins, which can therefore hardly be described as a single algorithm.

5.2.4 Application

The considered application is the GATE Monte-Carlo code implemented as a MOTEUR workflow available on the VIP/Gate-Lab web platform. As described in the previous chapter, the workflow mainly consists of (i) a computing part, when partial simulation results are produced by Monte-Carlo jobs, and (ii) a merging part, where these results are assembled together. The computing phase is implemented using the dynamic load-balancing method described in chapter 3, where each Monte-Carlo job contributes to the simulation uninterruptedly until it is stopped by the workflow engine. In these conditions, each computing resource executes at most one Monte-Carlo job. All jobs are single-core. Two execution modes are considered: with checkpointing and without checkpointing.

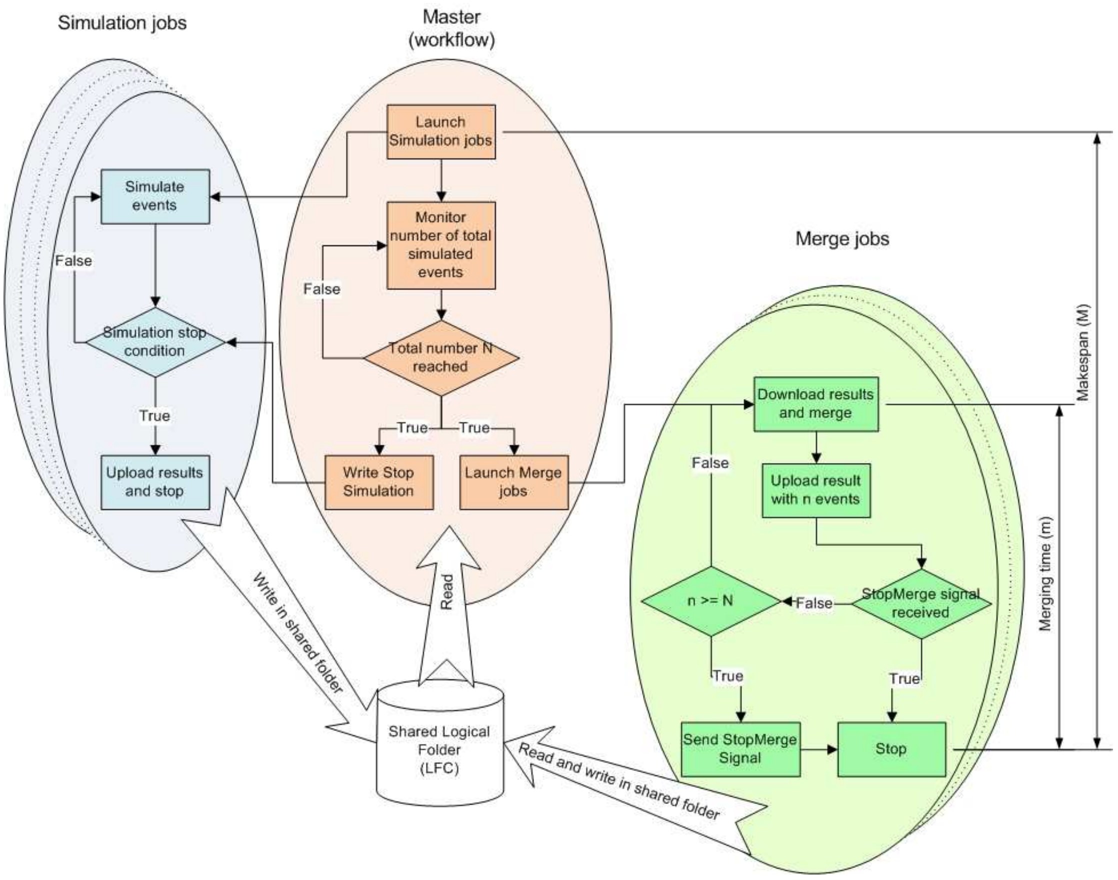


FIGURE 5.2: Monte-Carlo workflow without checkpointing (adapted from [Camarasu-Pop et al., 2013b]). Monte-Carlo jobs upload their results once at the end of their lifetime. The stop Monte-Carlo condition triggers: i) results upload from Monte-Carlo jobs and ii) the launch of merging jobs. The merge stop signal is sent by the first merging job having merged the required number of events. This figure is reproduced from chapter 4

Without checkpointing, merge jobs are launched once all the Monte-Carlo jobs are finished, while with checkpointing, they are launched from the beginning of the execution. In the latter case, merge jobs are active during the Monte-Carlo phase. To parallelize data transfers, several merge jobs can run concurrently. They rely on the central LFC to handle concurrency issues using logical directory creation/deletion as locking mechanism. The two application workflows, with and without checkpointing, are described in Figures 5.2 and 5.3, reproduced from Chapter 4.

5.3 The SimGrid toolkit

As discussed in Chapter 2, there exists a variety of simulation toolkits, among which OptorSim [Bell et al., 2003], GridSim [Buyya and Murshed, 2002], PeerSim [Montresor and Jelasity, 2009], CloudSim [Calheiros et al., 2011], and SimGrid [Casanova et al., 2008,

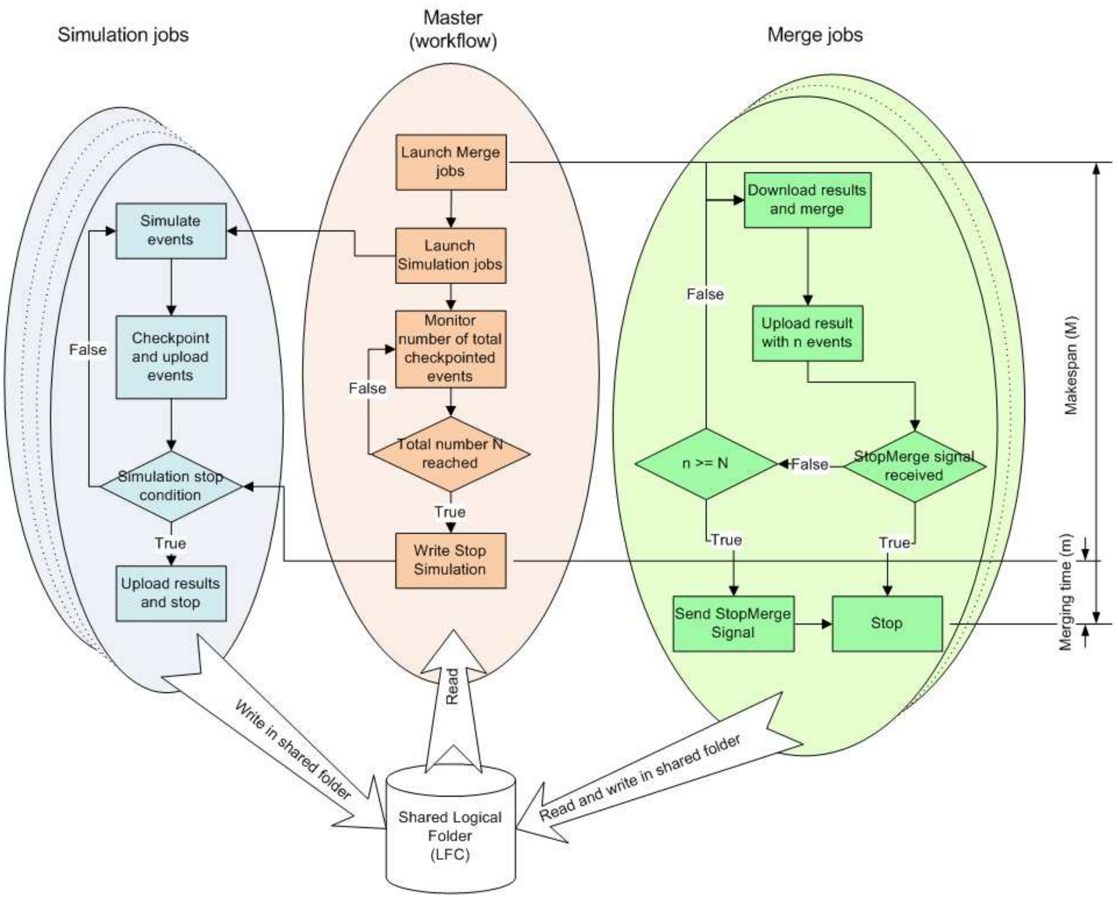


FIGURE 5.3: Monte-Carlo workflow with checkpointing (adapted from [Camarasu-Pop et al., 2013b]). Multiple Monte-Carlo and merge jobs are launched in parallel at the beginning of the workflow. Monte-Carlo jobs upload their results regularly, at the same frequency as they checkpoint their partial results. The stop Monte-Carlo condition given by the master triggers the end of Monte-Carlo jobs. From this moment on, only merging jobs continue to run (merging phase). The stopMerge signal is sent by the first merging job having merged the required number of events. This figure is reproduced from chapter 4

Bobelin et al., 2012]. Recent reviews of these tools are available, e.g., in [Beaumont et al., 2011] and in [Naicken et al., 2006] which focuses on peer-to-peer platforms. We chose to use the SimGrid toolkit because it is an active software project on which consistent efforts have been devoted to scalability, and which exposes a rich simulation toolkit through well-documented, easy and complete APIs. SimGrid provides core functionalities for the simulation of applications in heterogeneous distributed environments. In the following, we describe the functionalities used in our simulation.

5.3.1 Platform

A “SimGrid platform” consists of a hierarchical description of Autonomous Systems (AS). Each AS can contain (i) hosts, (ii) routers, (iii) links defining the connection between two (or more) resources (links have a bandwidth and a latency) and (iv) clusters, which can contain a certain number of hosts interconnected by some dedicated network. The routing between the elements of an AS has to be explicitly defined. Network is simulated using a fluid network model [Bobelin et al., 2012]. A “SimGrid host” defines a physical resource with computing capabilities, one or more mailboxes to enable communication, and some private data that can be only accessed by local processes. A host is identified by its id and peak power expressed in flop/s. An availability value can also be defined for the host along time; it expresses the percentage of computing power available. The ON/OFF state of the host can also be described along time. More information about platform description in SimGrid is available in [Frincu et al., 2008].

5.3.2 Services

Transfer and execution of “SimGrid tasks” are provided by SimGrid APIs. “SimGrid tasks” are defined by a computing amount, a message size and some private data. Tasks can account for both processing and data transfer (potentially with null computing amount). Task exchanges are handled like messages that can be sent and received from/to mailboxes. Once received, tasks are handled by *processes*, that may compute them or transfer them to others hosts. SimGrid offers several interfaces, among which MSG (Meta-SimGrid) for multi-agents simulations, SMPI for MPI simulations, and SimDag for Directed Acyclic Graphs (DAGs) of parallel tasks.

5.3.3 Deployment

Deployment defines on which SimGrid hosts the simulator deploys the simulated processes. It can be programmed using specific functions, or defined in a XML file which associates processes to hosts and specifies their arguments, for instance the list of workers of a master, or the site storage element.

5.4 Simulation Design

Table 5.1 compares the real and simulated systems. The simulation design is detailed hereafter.

5. Computed as running jobs plus free job slots from http://gstat.egi.eu/gstat/vo/biomed/

Function		Real system	Simulated system
Application description		MOTEUR workflow with parameters: – checkpointing period – number of mergers – number of Monte-Carlo jobs – number of Monte-Carlo events	Master and worker processes with parameters: – checkpointing period – number of mergers – number of Monte-Carlo jobs – number of Monte-Carlo events – computational cost of Monte-Carlo event * – computational cost of merge operation * – file size *
Services	File catalog	LFC	LFC process
	File transfer	SRM and gridFTP	SE process
	Job scheduling	MOTEUR and DIRAC	Master process
	Workload	Pilot and application jobs	Worker process
Deployment		gLite WMS and batch queues	Random selection of hosts for worker processes Latencies and failure times matched to real trace
Platform		EGI, biomed VO, 100 sites, 179 clusters, approx. 24,300 nodes[5] non-dedicated network	Grid'5000 platform model 10 sites, 40 clusters, approx. 1,500 nodes dedicated network

TABLE 5.1: Overview of real and simulated systems. * calibrated parameters

5.4.1 Platform

Despite various attempts to model the EGI platform, there is currently no suitable model available for our simulations. This is mainly due to the lack of detailed information on (i) the configuration (number and performance of processing units, intra-site bandwidth, etc.) of its more than 300 distributed sites and (ii) the inter-site network connections. Gathering the necessary information is particularly challenging because it changes frequently.

Consequently, we used the Grid5000 platform[6] model. It comprises 10 sites (AS), 40 clusters, approximately 1500 nodes, and the network infrastructure of the Grid5000 platform. We added a single-host cluster to each site of the platform to deploy its storage element; the network link used to connect this cluster had similar performance to the other links in the site.

5.4.2 Services

Two SimGrid processes were implemented to simulate (i) the LFC and (ii) an SE composed of an SRM and gridFTP service. They simulate communications involved in file transfer operations using the SimGrid MSG API. The main messages handled by our simulated LFC are file registration, directory management, and file replica listing. Only a single replica per file can be handled at the moment. Our SE has two operations: file upload and file download.

File transfers are implemented as tasks with null computing amount sent from the transfer source to the destination. I/O operations on the storage disks are assumed negligible with respect to cross-site transfers. Two methods were defined to simulate the gLite file transfer clients: (i) copy and register file, which consists of file upload to SE and registration in LFC, and (ii) copy file, which consists of file replica listing in LFC and file download from SE.

A master and a worker process were implemented to simulate job creation, scheduling and workload execution as illustrated on Figure 5.4. The master simulates both the MOTEUR workflow engine and the DIRAC scheduler. It first initializes the workers by sending them an `init` task with null computing amount and null data transfer. It then generates the Monte-Carlo jobs and sends them to worker processes as soon as they acknowledge the `init` task (first come, first served). If checkpointing is enabled, the master sends merge jobs to the workers from the beginning of the simulation; otherwise, it waits for the Monte-Carlo jobs to be finished.

6. https://www.grid5000.fr

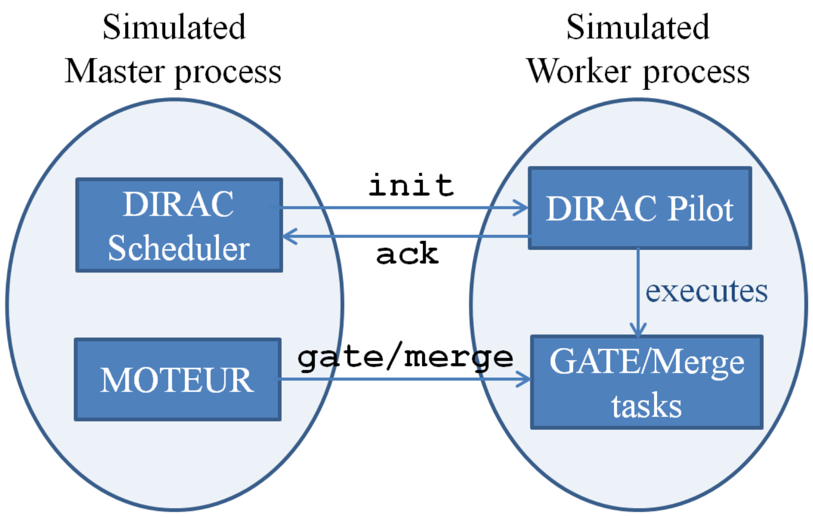


Figure 5.4: Master and worker simulated processes.

Worker processes simulate both DIRAC pilot jobs and the application jobs. They declare themselves to the master by sending an `ACK` message in response to `init`. They can process both Monte-Carlo and merge jobs. All processes are implemented using SimGrid's MSG API.

5.4.3 Deployment

The resource provisioning process, implemented by the gLite WMS, site batch queues and DIRAC pilot submitter in the real system, is simulated by randomly generating the list of hosts used to deploy workers for each simulation. Two separate host lists are used, one for each type of application worker. The master and LFC processes are deployed once and for all on random nodes in the platform.

Job latencies and failures are simulated using SimGrid's host availability and state files. This is possible because each platform host executes at most one Monte-Carlo job (see Section 5.2). A job latency of t is modeled by a host availability of 0 until time t. A job failure at time t is modeled by a host going from state ON to OFF at time t. Latency values and failure times are extracted from real traces. Table 5.2 summarizes simulation host properties and implementation details. To match latency/failure times extracted from real jobs to availability/state values of simulated hosts, we do a pairwise match between the list of real jobs sorted by decreasing values of measured CPU power and the list of simulated hosts sorted by decreasing values of assigned CPU power as illustrated on Table 5.3. This matching is done separately for each type of application worker. Job resubmissions triggered by MOTEUR when jobs fail are simulated as any other job.

Simulation host	Description file	Possible values	Source values
Power	Platform file	4.E9 - 30E9 flops/s	Grid5000 platform description
Latency	Availability file	0 until time t, 1 afterwards	Latency t extracted from real trace file
Failure	State file	ON until time t, OFF afterwards	Failure time t extracted from real trace file

Table 5.2: Simulation host properties and implementation details. Simulation hosts retrieve their CPU power from the Grid5000 platform description file. Job latencies and failures are extracted from real traces and are simulated using SimGrid's host availability and state files.

Host Number	Grid5000 Power (flops/s)	EGI Power (BogoMips)	Latency (s)	Failure (s)
1	23E9	4522	1440	240
2	27E9 ↓	5332 ↓	840	18000
3	30E9	5600	900	120

Table 5.3: Real and simulation hosts are sorted according to their CPU power. Examples of real job latencies and failure times are extracted from real execution logs and assigned to corresponding simulation jobs.

5.4.4 Application

Two types of application workers are simulated, one for Monte-Carlo jobs and one for merge jobs. Monte-Carlo jobs download their input data using the file transfer method, and start the Monte-Carlo simulation. They periodically report their number of computed events to the master by sending `events` messages. If checkpointing is enabled, they also upload their results to the site's SE using the file transfer method. Monte-Carlo jobs terminate and upload their final result when they receive a `stop` message from the master. A fixed file size is used for all Monte-Carlo results.

To handle concurrency among mergers, an atomic operation is implemented in the LFC process to return a unique list of files to merge at each invocation. Mergers invoke this operation, and download the returned files. The merging process itself is simulated by sleeping during the CPU time of the merging operation. The merged result is finally uploaded using the file transfer method. In case it contains more than the total number of events to compute, a `stopMerge` directory is written in the LFC. Merge processes periodically check the presence of this directory and terminate accordingly.

The master receives `events` messages specifying the number of Monte-Carlo events computed by the emitter. Based on these, it maintains the total number of computed events, and sends a `stop` message to all Monte-Carlo processes when all the events are computed.

5.4.5 Calibration

Calibration of application parameters is required to address performance discrepancies between the real and simulated platforms. We use the following rule to determine the computing cost of a Monte-Carlo event:

$$x[flop/event] = \frac{\bar{p}_{sim}[flop/s]CPU_{real}[s]}{E_{real}[event]}, \tag{5.1}$$

where the units are given in square brackets, x is the computational cost of an event, $\bar{p}_{sim}$ is the average performance of hosts on the simulation platform, CPU_{real} is the cumulative CPU time of the Monte-Carlo computation on the real platform, and E_{real} is the number of Monte-Carlo events in the real application. We consider real executions of a GATE Monte-Carlo computation of 50 million events which represents roughly 61,000 minutes of CPU on the biomed VO. The Grid'5000 platform model used for simulation has an average performance $\bar{p}$ of 12.68 Gflop/s, which for our calibrating application gives $x = 0.93$ Gflop/event.

We use a similar rule to determine the computing cost of a merge operation between two files:

$$y[flop/merge] = \bar{p}_{sim}[flop/s]CPUm_{real}[s], \tag{5.2}$$

where $CPUm_{real}$ is the CPU time of a merge operation on the real platform. We obtain $y = 126$ Gflop for our application and simulated platform.

The size of the files produced by Monte-Carlo jobs was calibrated so that the simulated file transfer times in merge jobs match best the median time measured on the real application. Figure 5.5 compares file transfer times in the merge jobs for the real and simulated experiments. Three file sizes, 10MB, 15MB and 20MB were tested. Based on this, we chose a file size of 15MB, which has a median transfer time very close to the real one for download (34s versus 35s) and reasonably close for upload (26s versus 18s).

5.5 Simulation validation

We made some experiments to compare the real and simulated execution time for different numbers of mergers and different checkpointing periods. For each experiment, the Monte-Carlo phase was executed by 300 parallel jobs. Table 5.4 summarizes the tested application configurations. Three real traces are available for each configuration, and 5 simulations were performed for each real trace. For a given real trace, simulations only differ by the random list of hosts used to deploy the application workers.

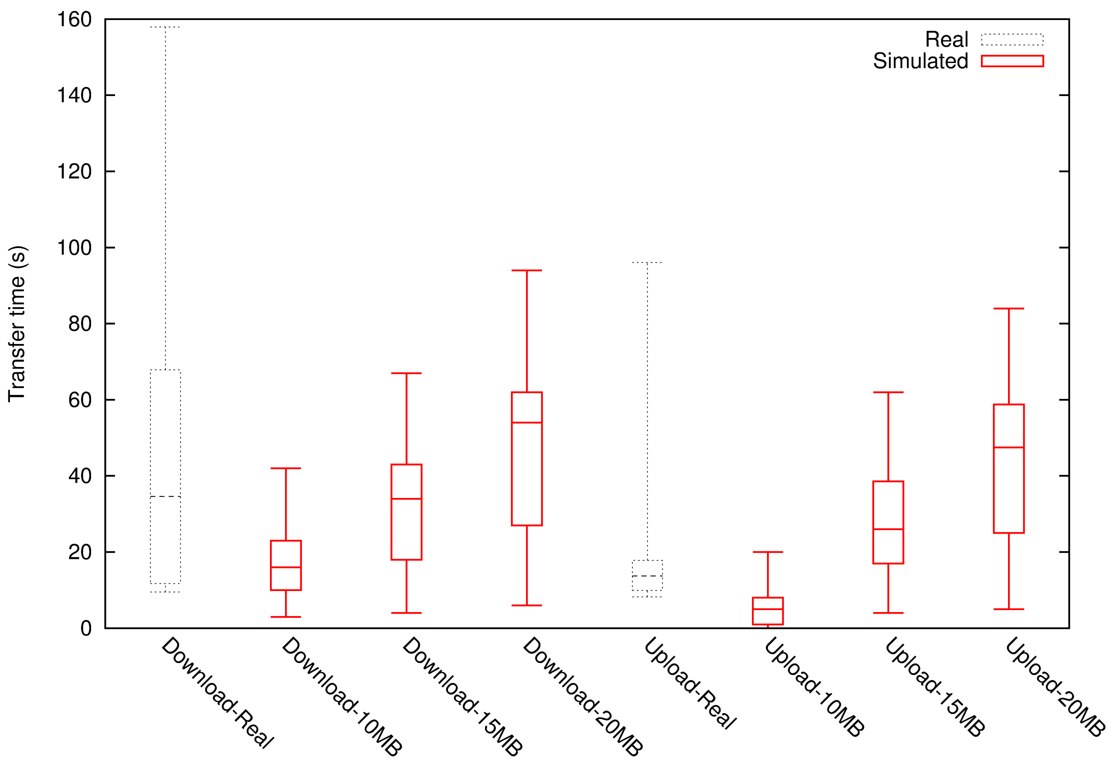


FIGURE 5.5: Calibration of the file size. A file size of 15MB was chosen.

5.5.1 Study of the Monte-Carlo phase

5.5.1.1 No checkpointing

Figure 5.6 shows an XY plot of the makespan of the Monte-Carlo phase in the simulated and real systems for the workflow without checkpointing. Results show a good match between the values of the simulation and the real experiments. They follow the same trend, but are slightly shifted, having an average relative error of 10%.

Nevertheless, we notice that the computing time is systematically under-estimated by the simulator. This is mainly due to an under-estimation of the real total CPU time, as explained further, in Section 5.7. The simulation results on the upper right-hand

# mergers	checkpointing period
1	disabled
5	disabled
10	disabled
15	disabled
10	30min
10	60min
10	120min

TABLE 5.4: Application configurations tested in the experiments.

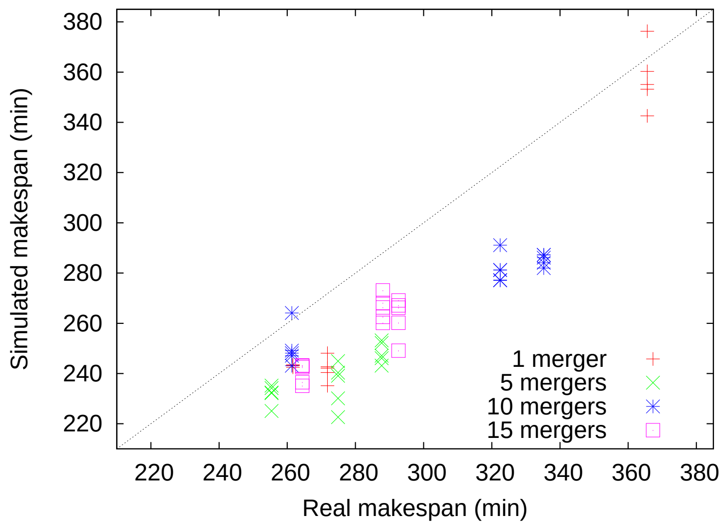


FIGURE 5.6: Real versus simulated makespans of the Monte-Carlo phase for the workflow without checkpointing, and the 4 merger configurations.

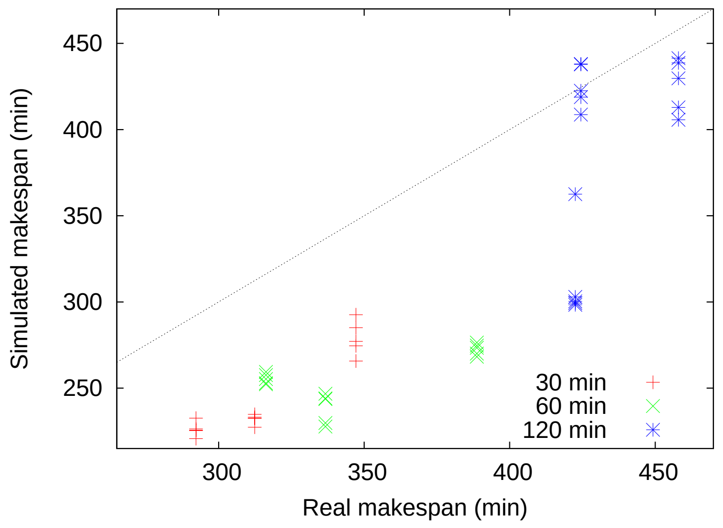


FIGURE 5.7: Real versus simulated makespans of the Monte-Carlo phase for the workflow with checkpointing (10 mergers).

side corner of Figure 5.6 are the ones that best match their real trace. This trace corresponds to a real execution with a very important latency, i.e. an average latency of approximately 90 min w.r.t latencies of 25-50 min for the other executions. The latency being a measured parameter directly injected in the simulator, the simulated makespan is thus closer to the real makespan values than for the other executions.

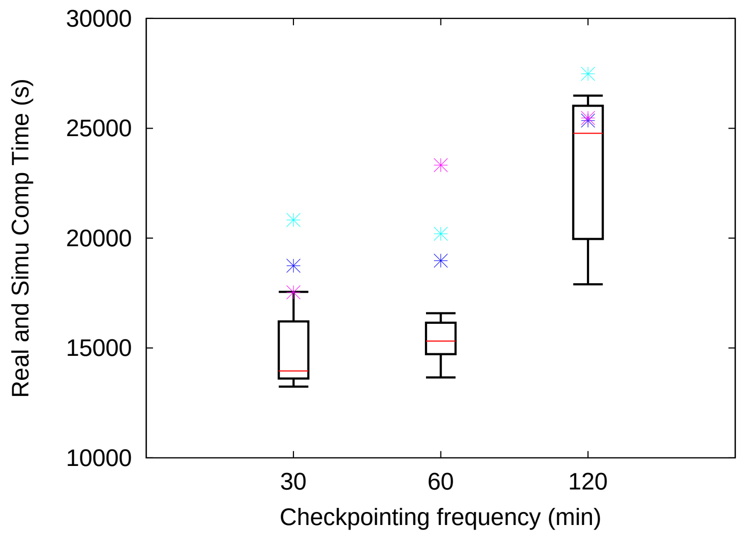


FIGURE 5.8: Real and simulated makespans of the Monte-Carlo phase as a function of the checkpointing period for the workflow with checkpointing. The 9 real values are shown individually, while the 45 simulated values are grouped in three box-and-whisker plots with median values, one for each checkpointing frequency.

5.5.1.2 With checkpointing

Figure 5.7 shows an XY plot of the makespan of the Monte-Carlo phase in the simulated and real systems for the workflow with checkpointing. They follow the same trend and the match is still good, but the relative error is larger, with an average of 20%. Overall, we notice the same under-estimation as for the workflow without checkpointing (Figure 5.6).

Figure 5.8 shows real and simulated computing times as a function of the checkpointing period. We notice that the simulation and real values follow the same trend: the makespan increases with the checkpointing period, which is consistent with the observations in Chapter 4.

Simulation results with a checkpointing period of 120 min seem to be more accurate than the others (Figures 5.7 and 5.8). This is because two among the three real executions had a very important latency which was directly injected in the simulator.

5.5.2 Study of the merging phase

Figure 5.9 shows the real and simulated makespans of the merging phase as a function of the number of mergers for the workflow without checkpointing. The simulated and real makespans follow the same trend: merging time decreases when the number of parallel mergers increases, until the threshold of 10 mergers after which it slightly increases.

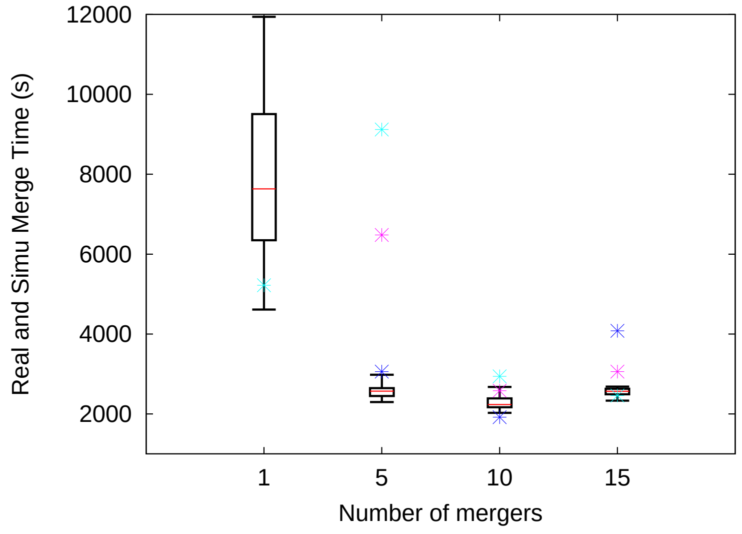


FIGURE 5.9: Real and simulated makespans of the merging phase as a function of the number of mergers for the workflow without checkpointing. The 12 real values are shown individually, while the 60 simulated values are regrouped in four box-and-whisker plots with median values, one for each number of mergers. Due to scale issues, only 1 real value is displayed for 1 merger; the two others are at 19980 s and 26760 s.

The simulated makespan is less variable than the real one for one and five mergers. This is most likely because the Grid5000 platform used in the simulation is significantly less heterogeneous that the EGI used for the real executions. The performance variability coming from the random choice of hosts in the simulated deployment remains lower than the variability encountered in reality.

When there is only one merger, the merging time is tightly dependent on the performance of the selected node. This explains the higher variability of the makespan for one merger. When the number of mergers increases (10 or 15), the merging performance is less sensitive to heterogeneity since a few good hosts will complete the work not accomplished by the slow ones. Results for the real executions run with the same parameters are thus closer. In these cases the simulation matches very well the real observations.

Figure 5.10 shows the real and simulated makespans of the merging phase as a function of the checkpointing period for the workflow with checkpointing. Simulation results are close to real values and follow the same trend: merging time decreases when the checkpointing period increases because the number of partial results to merge decreases. For a checkpointing period of 30 minutes, the performance of the real application is very variable. This is a borderline case (as was the workflow without checkpointing and one merger) for which platform heterogeneity plays an important role. The simulation manages to capture these differences only to a certain extent, simulation values showing more variability than for higher checkpointing periods.

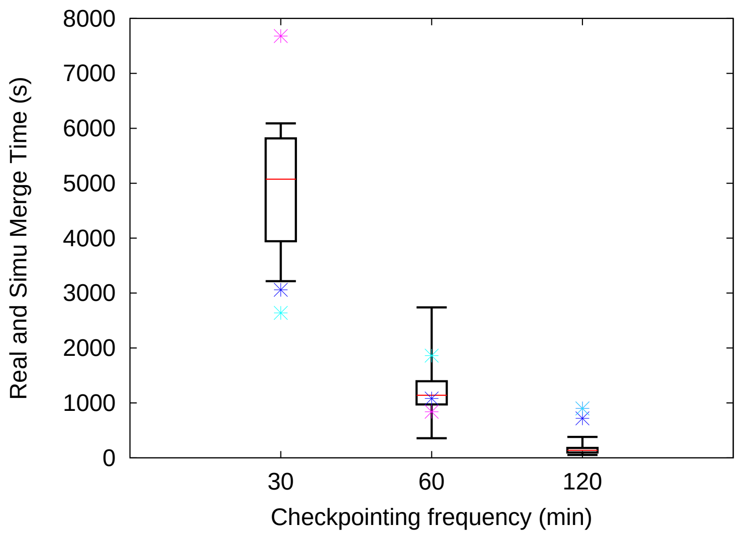


FIGURE 5.10: Real and simulated makespans of the merging phase as a function of the checkpointing period for the workflow with checkpointing. Real values are shown individually, while the 45 simulation values are grouped in three box-and-whisker plots with median values, one for each checkpointing period.

5.6 Parameter studies

The previous section showed that simulation results are consistent with real values. Simulation can be thus used to further investigate the influence of certain parameters on the application performance. The checkpointing period, for instance, was previously tested with only three different values. Simulation enables a more exhaustive study. In the following we will present results obtained with eleven different checkpointing periods varying from 20 minutes to two hours.

Figures 5.11, 5.12, 5.13 show computing, merging and total makespans obtained in simulation as a function of the checkpointing period with the following values : 20min, 25min, 30min, 35min, 40min, 45min, 60min, 75min 90min, 105min and 120min. Each graph uses one trace corresponding to one real execution with checkpointing. In total, 3 traces are presented; the 6 others are available in the appendix A. For each trace we executed 55 simulations, i.e. 5 repetitions (each with a different simulation deployment file) of each of the 11 checkpointing periods.

The simulated merging makespan gives a valuable overview of the merging performance with respect to the checkpointing period. We can clearly see that the merging time decreases when the checkpointing period increases up to a threshold that can be estimated at 45 minutes. The total makespan confirms that this value gives the best results for the complete execution. The simulation helps in finding the right checkpointing period at a very low cost compared to the real executions (for which we had not tested this

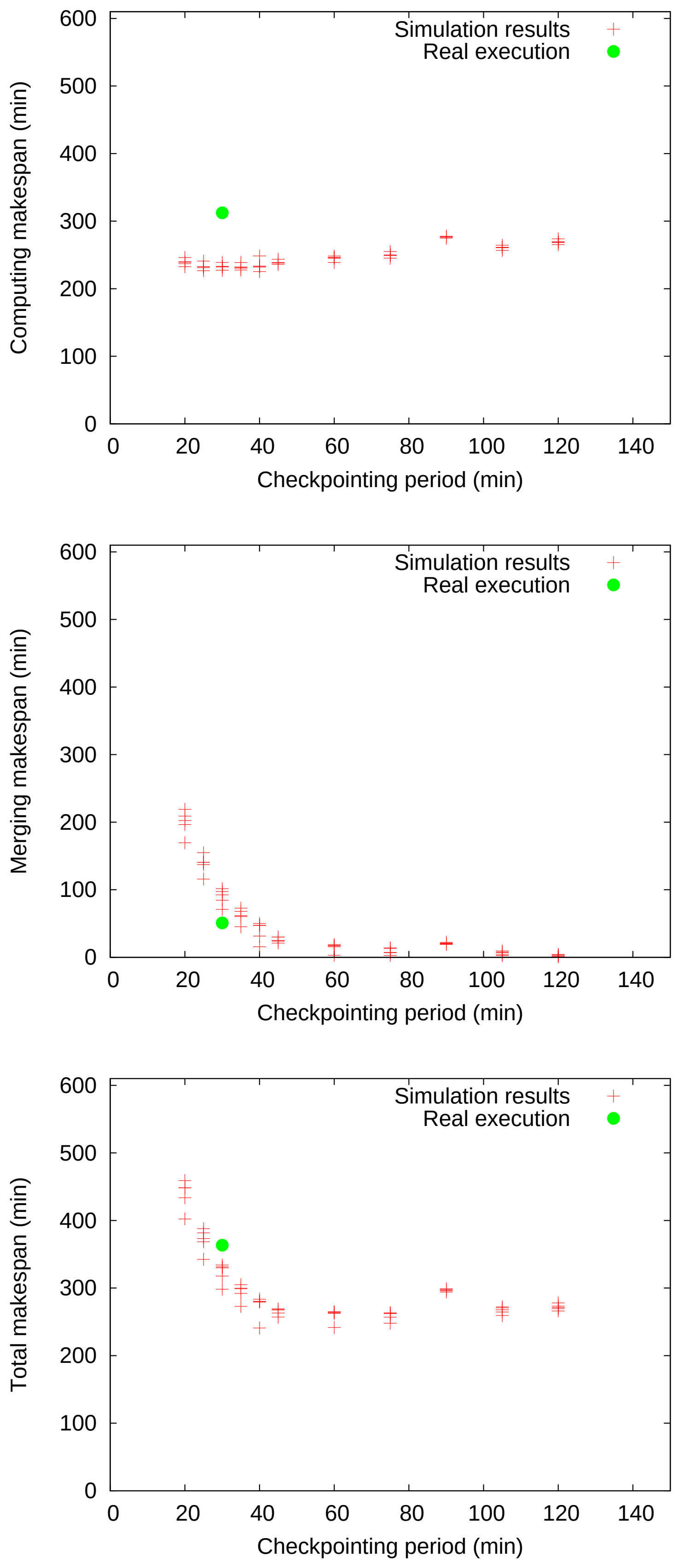


FIGURE 5.11: Computing, merging and total makespans as a function of the checkpointing period. The real execution has a checkpointing period of 30 minutes and corresponds to trace 1, used as input for the simulated results.

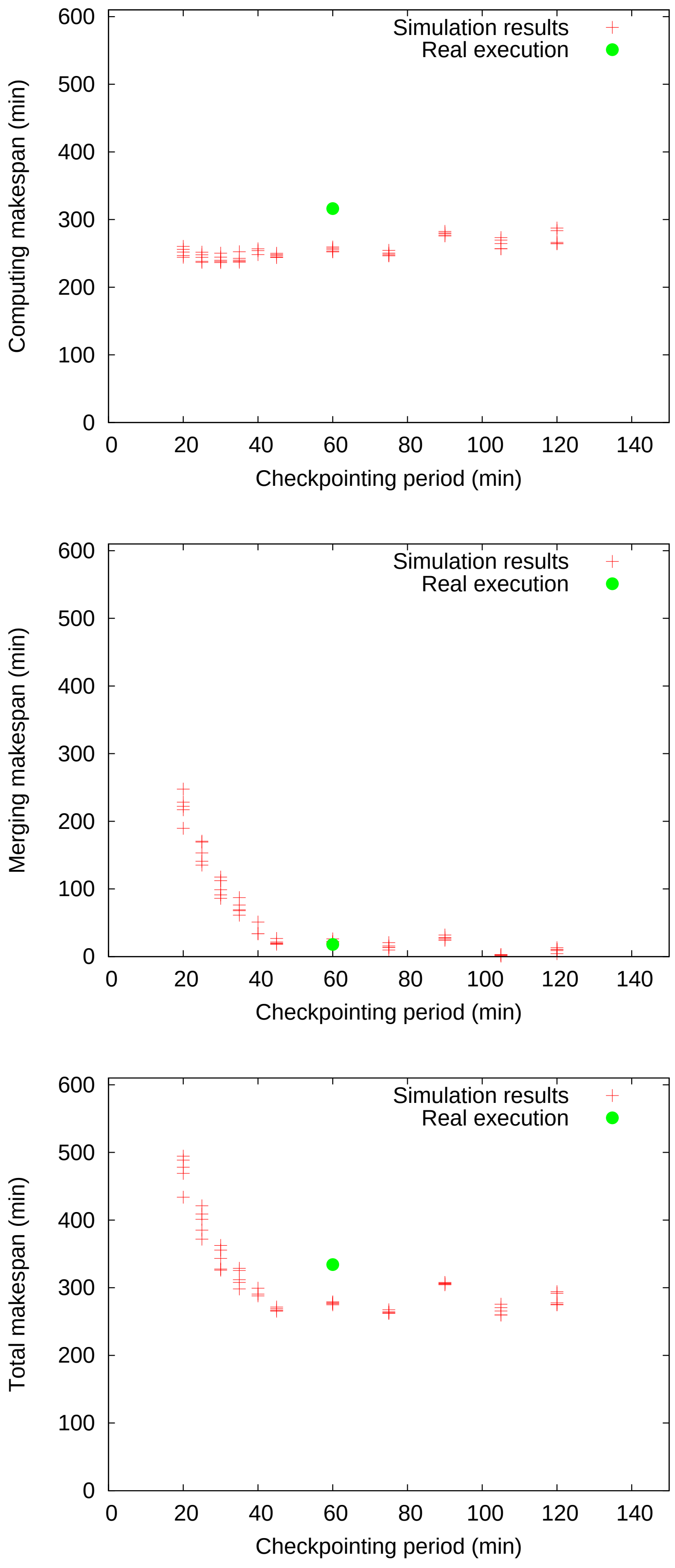


FIGURE 5.12: Computing, merging and total makespans as a function of the checkpointing period. The real execution has a checkpointing period of 60 minutes and corresponds to trace 4, used as input for the simulated results.

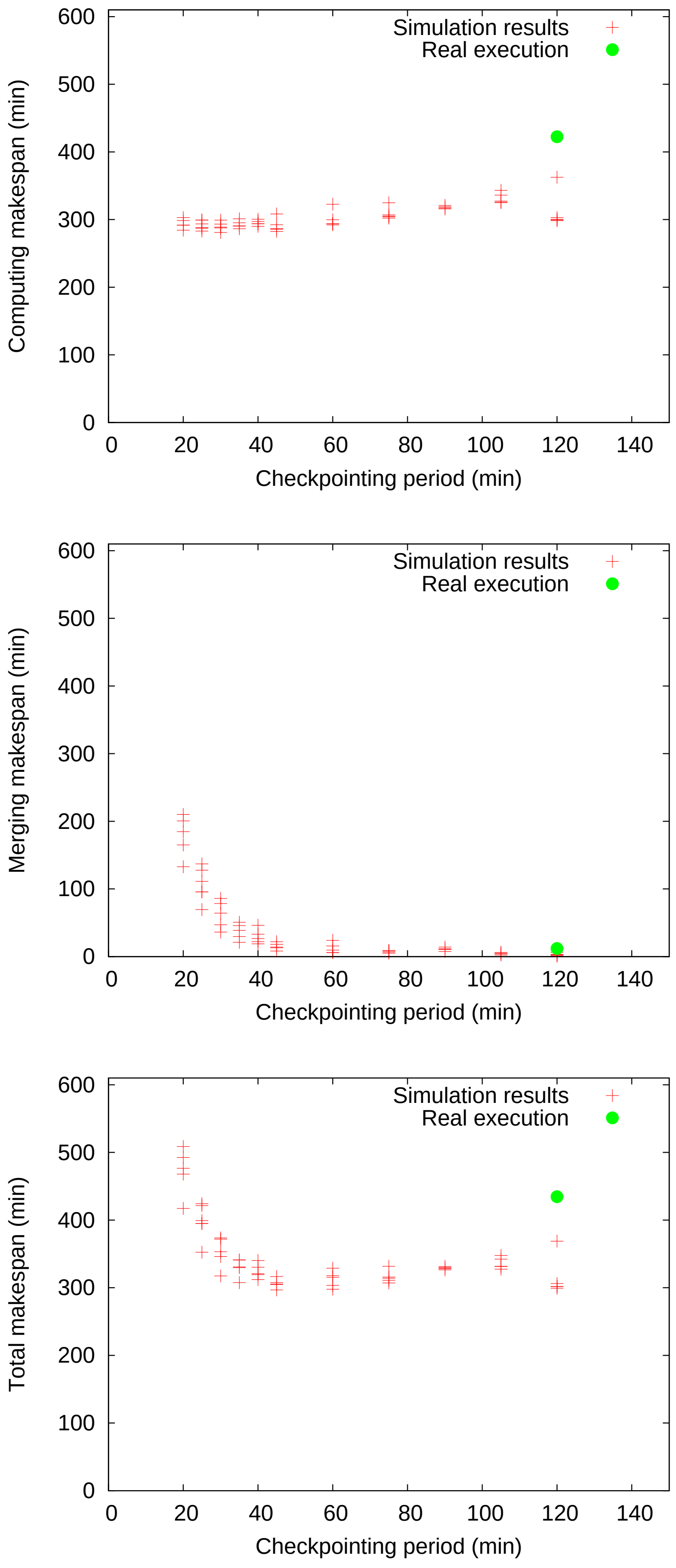


FIGURE 5.13: Computing, merging and total makespans as a function of the checkpointing period. The real execution has a checkpointing period of two hours and corresponds to trace 7, used as input for the simulated results.

value). In this case, 45 minutes is the lowest acceptable threshold for the checkpointing period. Below this value, the mergers are overcharged leading to very high merging times. Above this value, checkpointing overhead is limited and checkpointing can be used to increase reliability as discussed in chapter 4. Therefore, we conclude that, when choosing a checkpointing period, it is safer to overestimate it than to underestimate it.

Simulation results presented in Figures 5.11, 5.12, 5.13 can be used beyond parameter tuning. Curves obtained for the merging makespan seem to follow an exponential decrease. This opens the way to modeling the merging makespan with the help of a decreasing exponential function, which would enrich the model proposed in chapter 3 where the merging makespan was only described by a measured parameter m.

5.7 Discussion

The good match between real and simulated performance can be explained by the realism of SimGrid, by our calibration of the application on the simulated platform, and by the injection of real latency and failure values in the simulation. Nevertheless, the calibration phase and the injection of traces could still be improved.

First, some of the traces used here were incomplete due to canceled jobs. Indeed, jobs that do not answer the `stop` signal within a given delay are canceled and their traces are lost. These are isolated cases for the workflow without checkpointing, but rather frequent for the workflow with checkpointing. Missing traces involve lack of data data on (i) the real node power and on (ii) the total CPU used. Thus, the total real CPU time could be under-estimated, which would explain why simulation results for the computing phase show a lower makespan compared to real values.

Second, the computing cost of a Monte-Carlo event was computed based on the average power of all real executions and on the average power of the entire Grid5000 platform. We could improve the accuracy of the computing cost by assigning host power based on real values (extracted from real traces) to the Grid5000 simulation hosts. For a given trace (corresponding to one real execution) we could compute its average real power $\overline{c}$. Then, for each simulation host i, its computing power would be $p_i = \frac{c_i}{\overline{c}} * \bar{p}$, where $\bar{p}$ is the average Grid5000 power.

Overall, our results show that realistic simulated behaviors may be obtained on a simulated platform that corresponds to the real one only to a limited extent. We believe that this is an interesting conclusion given the difficulty to build realistic models of real large-scale platforms. Nevertheless, further investigations are needed to determine to what extent (in which conditions) a simulated platform can be used as a model of a real platform.

5.8 Conclusion

We presented a simulation framework for a Monte-Carlo application workflow deployed on EGI with the MOTEUR engine and DIRAC pilot jobs accessing data on storage elements via the LFC file catalog. The considered application and deployment are complex and similar to many deployments in EGI. Based on the SimGrid toolkit, we simulated the deployment of pilot jobs by randomly selecting hosts and matching latencies and failure times to real traces. Job scheduling, application jobs and the main EGI data management services (file catalog and storage element) were also simulated as SimGrid processes. To address performance discrepancies between the real and simulated platforms, the computational cost and file size of the application were calibrated from real traces.

The resulting simulator was used to study the influence of the number of mergers in the application workflow, and its checkpointing period. Results show that absolute makespan values of the Monte-Carlo phase are consistent, and that the trends observed in production are correctly reproduced by the simulation: (1) increasing the checkpointing period increases the makespan of the Monte-Carlo phase, and it decreases the makespan of the merge phase, and (2) increasing the number of mergers decreases the makespan of the merging phase up to a threshold beyond which it increases again. In addition, the simulation reveals that the performance of the merging phase is highly impacted by the heterogeneity of the platform, which could hardly be observed in production.

We therefore conclude that SimGrid can be used to simulate performance studies of applications running on EGI. In view of the technical expertise, time and human cost required to perform meaningful performance studies in production conditions, this opens the door to much better and faster evaluations for applications deployed on this platform.

Moreover, parameter-studies results brought deeper insight w.r.t. previous observation on the real system. Indeed, curves obtained for the merging makespan could be used beyond parameter tuning, since they could allow the fitting of mathematical models on the merging makespan.

The objective of replaying in simulation a Monte-Carlo computation executed with dynamic load-balancing on EGI using traces extracted from the Virtual Imaging Platform is fulfilled. However, much remains to be done to replay *any* workflow execution. For Monte-Carlo applications, a static load balancing scheme, where tasks compute a fixed number of events, should also be implemented since it corresponds to the most used configuration (although sub-optimal). In addition, a more elaborated workflow engine should be simulated to enable the simulation of any application available in the VIP/Gate-Lab web platform.

Chapter 6

Exploitation: the GATE-Lab

Abstract *The GATE-Lab project started in 2009 and aimed at providing a friendly and reliable platform allowing researchers to easily execute distributed GATE simulations, as described in [Camarasu-Pop et al., 2011]. The GATE-Lab was later integrated into the Virtual Imaging Platform[1] (VIP) presented in [Glatard et al., 2012]. The GATE-Lab counts today 300 users and completes more than 150 GATE executions per month. Its success as a production tool is due, on the one hand, to the friendly and easy to use interface, and on the other hand, to the specific optimizations presented in the chapters 3 and 4, which render it reliable and efficient. This chapter gives some usage statistics and a few examples of results obtained in production illustrating (i) that the GATE-Lab experience is significant in terms of users and executions, (ii) that the dynamic load balancing can be used extensively in production, and (iii) that it significantly improves performance regardless of the variable grid conditions.*

6.1 Introduction

The previous chapters presented our contributions to optimize the exploitation of distributed heterogeneous infrastructures for Monte-Carlo simulations in the medical field. We integrated these contributions into the GATE-Lab system, a tool that has both motivated these contributions and helped validate them in a production environment, beyond controlled experiments. The GATE-Lab is now part of the VIP web platform openly available to the community and the dynamic load-balancing algorithm is used daily by tens of GATE-Lab users.

As described in Chapter 2, a few different portals for Monte-Carlo applications provide convenient access to grid tools and services [Engh et al., 2003, Compostella et al.,

1. http://vip.creatis.insa-lyon.fr

2007, Josï¿½ Carlos Mourïï¿½o Gallego, 2007, Pena et al., 2009], but are often targeting specialized communities such as radiotherapists. The GATE-Lab targets mainly researchers, both in academic and R&D industrial environments. The end user in our case is not the medical physicist sending a treatment plan to be computed, but the researcher who provides a file of macros describing a simulation. Researchers need to easily test simulations with different sets of parameters, to perform computing intensive simulations or to study the design of new imaging devices. The GATE-Lab allows them to run CPU-intensive GATE executions easily and rapidly without being directly exposed to the computing and storage the computing resources provided by the European Grid Infrastructure.

The rest of the chapter is organized as follows: Section 6.2 describes the GATE-Lab system implementation, and Section 6.3 presents usage statistics and production results showing the good performance achieved daily on tens of GATE execution using the dynamic load balancing. The chapter closes on a discussion on the use of multiple mergers and checkpointing strategies, and conclusions.

6.2 Implementation

6.2.1 Architecture

The VIP platform provides online access to imaging simulators. It describes pipelines as MOTEUR workflows executed with the MOTEUR engine [Glatard et al., 2008a] using DIRAC pilot jobs. The overall architecture of the platform is diagrammed on Fig. 6.1 and corresponds to the main functionalities presented in Chapter 5, Section 5.2. In addition to GATE, some 10 different workflows are currently available for medical image simulators of Radio Therapy, Computed Tomography (CT), Positron Emission Tomography (PET) Magnetic Resonance Imaging (MRI) and Ultrasound imaging (US), as well as for neuroimaging applications such as FSL and Freesurfer.

The motivation for using workflows is twofold. At design time, it provides an accessible, high-level formal representation used to determine sub-processes that can be enacted concurrently on distributed computing platforms. At runtime, the workflow is interpreted and executed using the MOTEUR workflow engine, which provides an asynchronous grid-aware enactor. Workflows are described using GWENDIA, a language dedicated to data-intensive scientific applications description. The VIP simulation workflows are extensively described in [Marion et al., 2011].

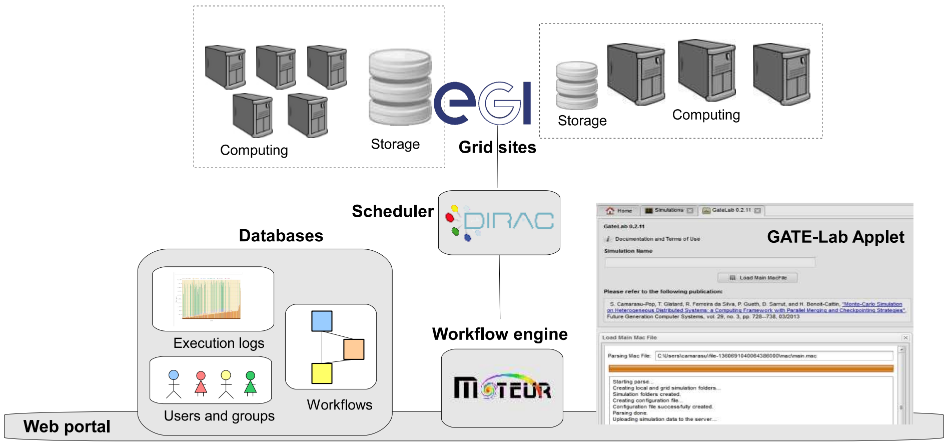


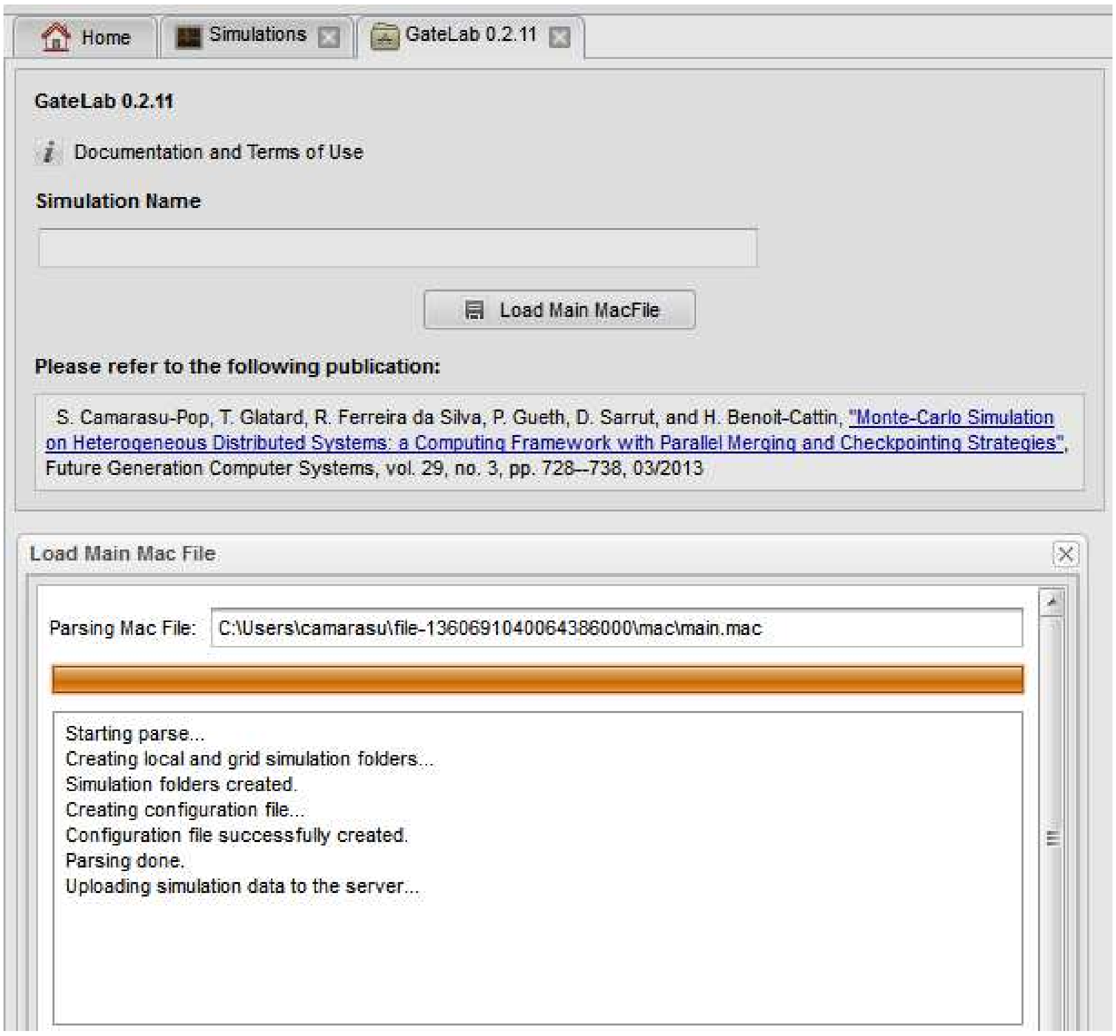
FIGURE 6.1: Architecture of the Virtual Imaging Platform.

The MOTEUR enactor is architected as a two-level engine. At the upper level, the core engine interprets the workflow representation, evaluates the resulting data flows and produces computational tasks ready for remote execution through a generic interface. At the lower level, the tasks descriptions are converted into jobs targeting a specific computing infrastructure. Jobs are submitted on the resources of the EGI biomed VO using DIRAC pilot jobs.

The DIRAC [Tsaregorodtsev et al., 2009] Workload Management System (WMS) implements a "pull" scheduling in which all the jobs are submitted to the central Task Queue and pilot jobs are sent to Worker Nodes at the same time. Pilot jobs run special agents that "pull" user jobs from the Task Queue, set up their environment and steer their execution. This approach has been presented more into detail in Chapter 2, Section 2.2.

6.2.2 GATE-Lab applet and interface

The VIP portal implements a client-server architecture based on servlets. It allows users to submit, monitor and manage their workflows. When the user selects an application, the portal generates a form from its workflow description. This form is filled in with input files and parameters and the workflow is submitted to MOTEUR. As illustrated on Figure 6.2, we developed a Java applet specifically for the GATE-Lab, to help the



GateLab 0.2.11

Documentation and Terms of Use

Simulation Name

Please refer to the following publication:

S. Camarasu-Pop, T. Glatard, R. Ferreira da Silva, P. Gueth, D. Sarrut, and H. Benoit-Cattin, "Monte-Carlo Simulation on Heterogeneous Distributed Systems: a Computing Framework with Parallel Merging and Checkpointing Strategies", Future Generation Computer Systems, vol. 29, no. 3, pp. 728--738, 03/2013

Load Main Mac File

Parsing Mac File: C:\Users\camarasu\file-1360691040064386000\mac\main.mac

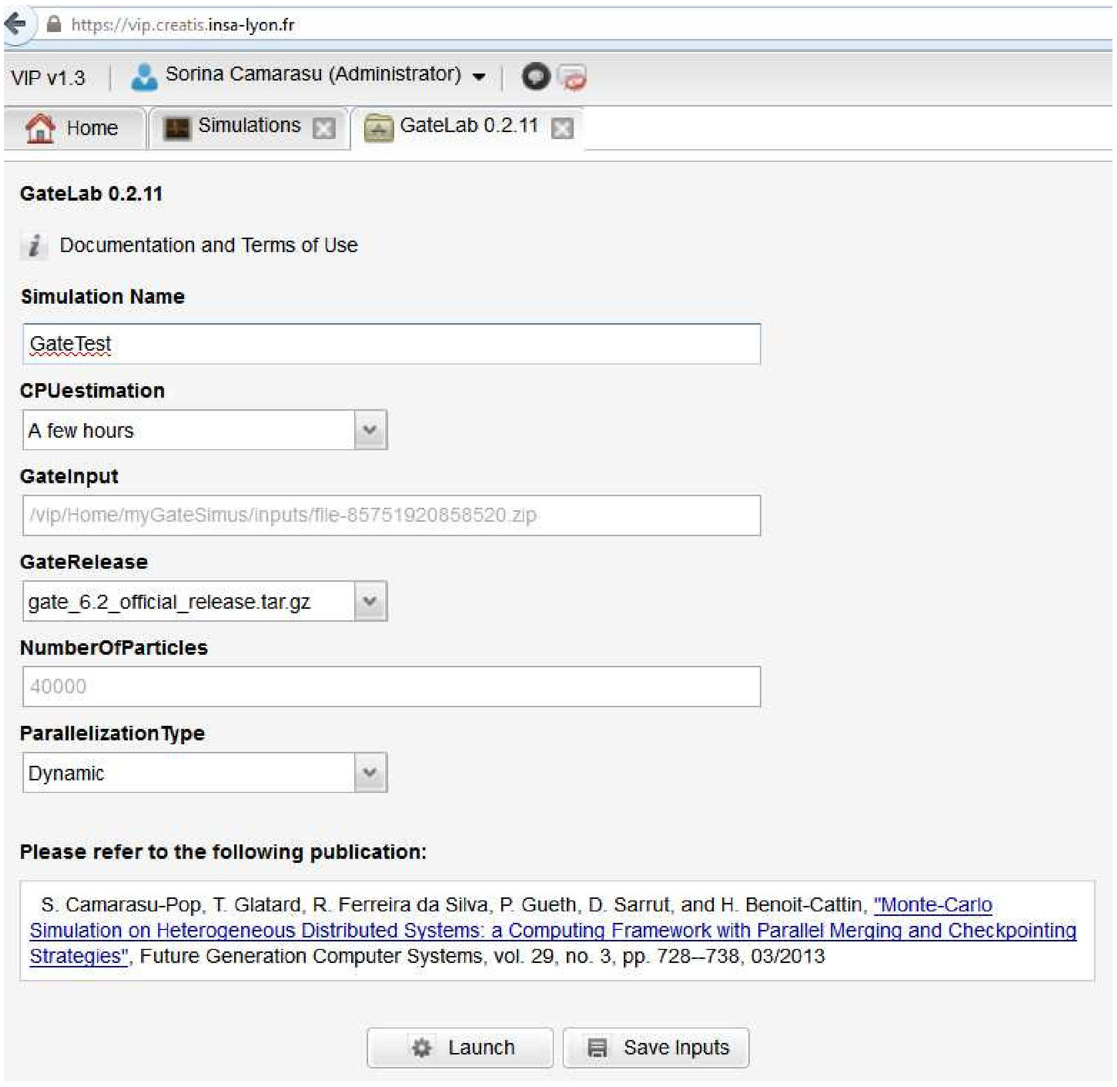
```
Starting parse...
Creating local and grid simulation folders...
Simulation folders created.
Creating configuration file...
Configuration file successfully created.
Parsing done.
Uploading simulation data to the server...
```

Figure 6.2: Screen capture of the GATE-Lab applet. The GATE-Lab applet looks for all local inputs files, bundles them into an archive and uploads it to the grid. The applet also checks simulation parameters and parses execution inputs.

user prepare his GATE simulation for its grid execution and automatically fill in some of the input parameters. An applet was needed in order to access input files directly on the user workstation, to parse them locally and upload them on the VIP server.

A GATE simulation is composed of several files, among which one or more text files that contain the macros describing the simulation, as well as data files such as images or database of materials description. These files are located on the user workstation and need to be transferred on the grid storage space in order to be available for the GATE execution on distributed worker nodes. Starting from the main macro file, the GATE-Lab applet looks for all local inputs files, bundles them into an archive and uploads it to the grid as illustrated on Figure 6.2.

The applet checks simulation parameters, in particular that the correct mode of the


https://vip.creatis.insa-lyon.fr

VIP v1.3 | Sorina Camarasu (Administrator)

Home | Simulations | GateLab 0.2.11

GateLab 0.2.11

Documentation and Terms of Use

Simulation Name

GateTest

CPUestimation

A few hours

GateInput

/vip/Home/myGateSimus/inputs/file-85751920858520.zip

GateRelease

gate_6.2_official_release.tar.gz

NumberOfParticles

40000

ParallelizationType

Dynamic

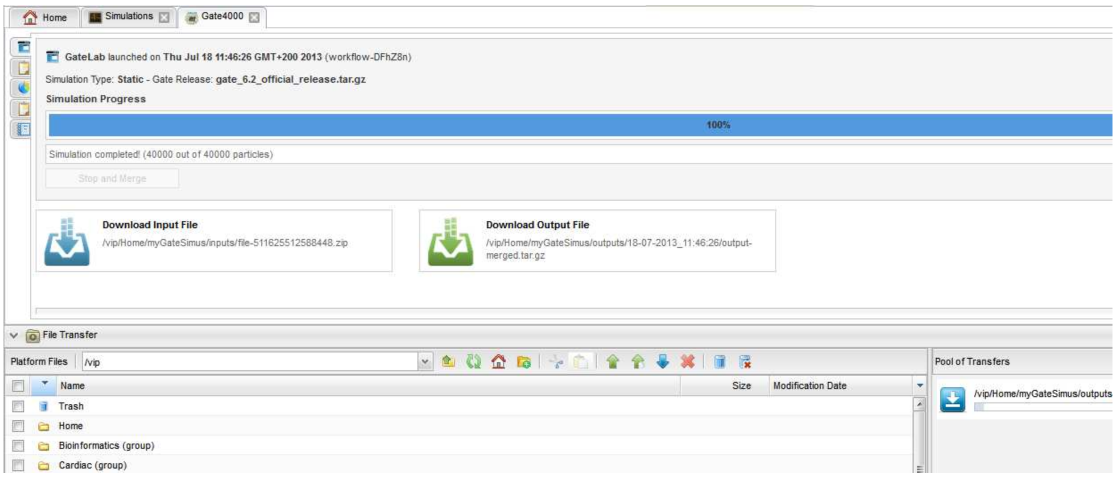
Please refer to the following publication:

S. Camarasu-Pop, T. Glatard, R. Ferreira da Silva, P. Gueth, D. Sarrut, and H. Benoit-Cattin, "Monte-Carlo Simulation on Heterogeneous Distributed Systems: a Computing Framework with Parallel Merging and Checkpointing Strategies", Future Generation Computer Systems, vol. 29, no. 3, pp. 728--738, 03/2013

FIGURE 6.3: GATE-Lab submit form. Inputs have been filled in automatically by the applet. The end user is asked for an estimation of the total CPU time of the GATE execution and for a name to be given to the current execution.

random engine generator is set so that each job uses a different seed. The applet also parses the total number of events to simulate and available timing information. Indeed, there are some commands (e.g. "setTimeSlice") indicating that the order of events in time is important and thus not compatible with the dynamic parallelization. In this case, only static parallelization is proposed to the user.

Once the applet finishes its uploads and parameter checks, the end user is asked for an estimation of the total CPU time of the simulation. Depending on this estimation, the simulation is split into a different number of tasks. By offering to the end user a limited number of choices expressed in terms of computing time, we simplify his job and we make sure that the number of tasks fits the workload reasonably well. The GATE-Lab integrates a static mapping between the time estimation given by the end user



GateLab launched on Thu Jul 18 11:46:26 GMT+200 2013 (workflow-DFhZ8n)

Simulation Type: Static - Gate Release: gate_6.2_official_release.tar.gz

Simulation Progress

100%

Simulation completed! (40000 out of 40000 particles)

Download Input File
/vip/Home/myGateSimus/inputs/file-511625512588448.zip

Download Output File
/vip/Home/myGateSimus/outputs/18-07-2013_11:46:26/output-merged.tar.gz

/vip

Name	Size	Modification Date
Trash		
Home		
Bioinformatics (group)		
Cardiac (group)		

/vip/Home/myGateSimus/outputs

Figure 6.4: Download results. When the GATE execution is finished, the user can download the final result with the file transfer pool.

and the number of tasks to submit. The mapping was decided based on our extensive experience with EGI, and correspond to a minimum of 5 jobs for short simulations (a few minutes) and goes up to a maximum of 500 jobs for simulations lasting more than a day. The final submit form is shown on Figure 6.3.

As illustrated on Figure 6.3, the user has the choice of the GATE release, among a list of releases that have been prepared for the GATE-Lab and previously uploaded on EGI storage elements. The default choice is the latest official GATE release [2], but older versions remain available for compatibility reasons.

Once the execution is finished, users can transfer output files between from the grid storage to their local machine by using a file transfer tool. File download is split in two steps: (i) transferring the file from the grid to the portal's server machine using the transfer pool and (ii) downloading the file to the user host. Figure 6.4 illustrates the user view of the execution, once it is finished.

The portal also provides statistics of workflow executions, jobs statuses and execution logs.

6.3 Production results

Contributions presented in the previous chapters have been evaluated independently through different sets of experiments. Though executed on EGI in production conditions, these experiments can be considered controlled, in the sense that they have been planned

2. hrefhttp://www.opengatecollaboration.org/releasedownload

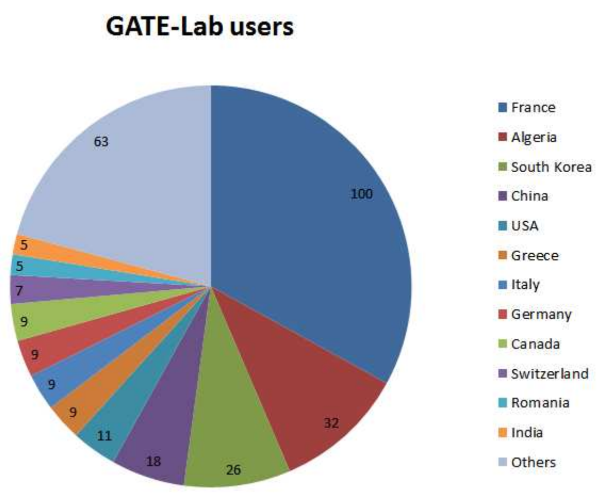


FIGURE 6.5: Registered GATE-Lab users per country.

and launched by the authors within the scope of evaluating the proposed algorithms and implementations. This sections presents results obtained by real GATE-Lab users, whose goal was to execute CPU-intensive GATE executions easily and rapidly without worrying about the underlying infrastructure, load-balancing techniques or other grid issues.

Results presented here aim at illustrating (i) that the GATE-Lab experience is significant in terms of users and executions, (ii) that the dynamic load balancing can be used extensively in production, and (iii) that it significantly improves performance regardless of the variable grid conditions.

6.3.1 GATE-Lab usage statistics

VIP counts, in July 2013, 437 users from 50 different countries, among which 70% (i.e. 304 users) registered as GATE-Lab users (when registering, users have the choice among different user groups, which give them access to different applications). Figure 6.5 illustrates the repartition by country of the 304 GATE-Lab users, which indicates that the platform has a significant number of users coming from various places.

Figure 6.6 shows the number of total and completed GATE-Lab executions since 2011. We notice their steady evolution, indicating that the platform activity has been sustained over the last two years. In practice, we notice that the activity never stops : there are on average 10 executions running every day.

We also notice that overall the completion rate (completed versus submitted executions) is rather low, below 50%. Most failures are usually due to user mistakes or data transfers. User mistakes are rather frequent, especially for new users. These mistakes are currently

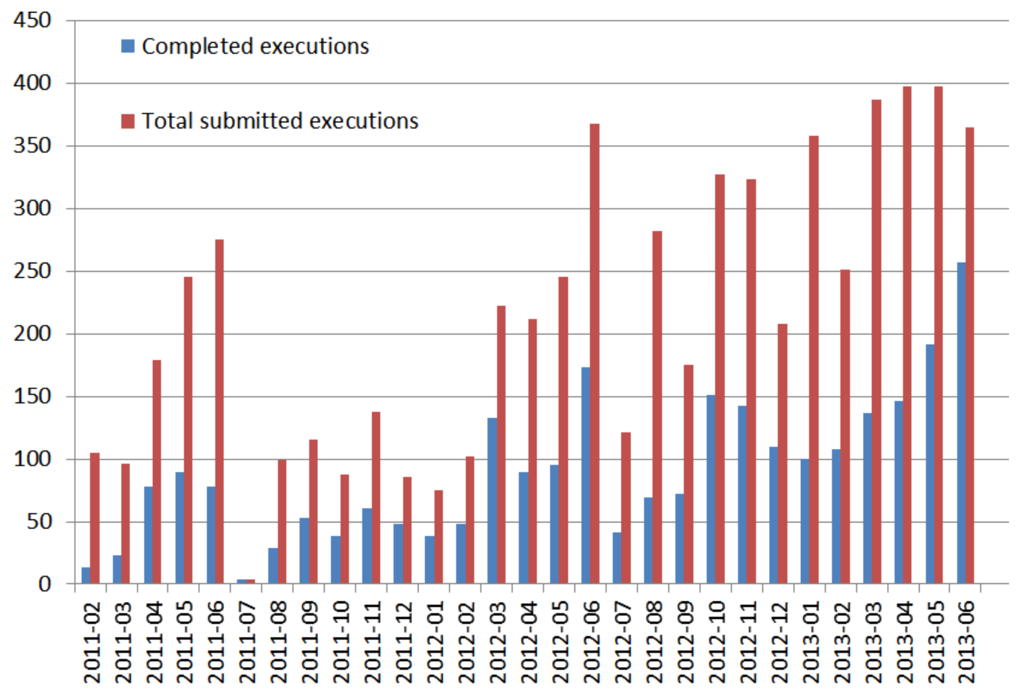


FIGURE 6.6: Total and completed GATE-Lab executions. Production activity is stable and growing, but error rate is still high.

detected by the users themselves or, more frequently, by the platform administrators. Thus their detection takes time and leads to numerous task failures. As future work, such failure cases should be better detected as presented in [Ferreira da Silva et al., 2013]. Data transfers are also an important issue. GATE simulations may have large input or output files that need to be transferred between the user workstation, the VIP/GATE-Lab platform and grid storage elements (SE). The GATE release can also be rather large (0.5 Go). Knowing that the concurrent jobs try to download the file simultaneously, this may lead to network congestion or SE overcharge. Ideally, each close SE should have its own replica to avoid cross-site transfers.

CPU usage is also significant. In June 2013, we had 259 completed (out of 372 submitted) GATE-Lab executions corresponding to 23 years of CPU time.

Among the total GATE executions launched with the GATE-Lab , there is a vast majority of dynamic executions. As illustrated in Table 6.1, during 6 months, from January 1st to June 30th 2013, there was a total of 2155 executions, among which 1553 dynamic executions and 602 static executions. During the same time interval, there were 939 completed executions, among which 852 dynamic executions and 87 static executions. We notice that there are considerably fewer static executions launched (less than one third) and even fewer completed. Overall we notice that (i) the completion rate of the static executions is much lower than for the dynamic ones and (ii) the number of static

GateLab executions	Completed	Total Submitted	Completion rate
Dynamic	852	1553	54%
Static	87	602	14%

Table 6.1: Dynamic vs Static completed and total submitted GATE-Lab executions from January 1st to June 30th 2013. Overall static executions have a much lower completion rate and are therefore less interesting for users.

executions launched represents less than one third (27%) of the total number of GATE-Lab executions. The dynamic mode is used whenever possible; as explained in Chapter 3, Section 3.4, the dynamic mode can only be used for Monte-Carlo applications for which all events are strictly equivalent and can be simulated in any order, which is currently not the case for all GATE executions.

In the following, we will give a few examples illustrating the way the dynamic executions adapt to different use cases and grid conditions.

6.3.2 Performance of the dynamic GATE executions using the GATE-Lab

As discussed in the previous section, the dynamic load-balancing approach is extensively used by the GATE-Lab. In Chapter 3, we compared the performance of the dynamic and static implementations and we showed that the dynamic mode significantly reduced the makespan by efficiently using the available resources until the end of the simulation. In the following we show additional performance results extracted from production executions launched by real GATE-Lab users. Our aim here is to illustrate the usability and efficiency of the proposed method on a production platform used by tens of users on a daily basis.

Figures 6.7, 6.8, 6.9 and 6.10, show four GateLab executions launched by real users in dynamic mode. Executions 1 and 2 were launched by *user*1 with 25 computing jobs

Property	Execution 1	Execution 2	Execution 3	Execution 4
Computing jobs	25	25	500	500
Speed-up[3]	12	21	255	392
Makespan (hours)	3.6	1.7	38.5	17.3
Cumulated CPU time (days)	1.9	1.5	411	282
Mean waiting time (min)	58	2	45	53
Error rate (%)	10	0	49	25

Table 6.2: Performance statistics for the executions plotted on Figures 6.7, 6.8, Figures 6.9 and 6.10.

3. The speed-up is defined as the the cumulated CPU time of the computing jobs divided by the makespan

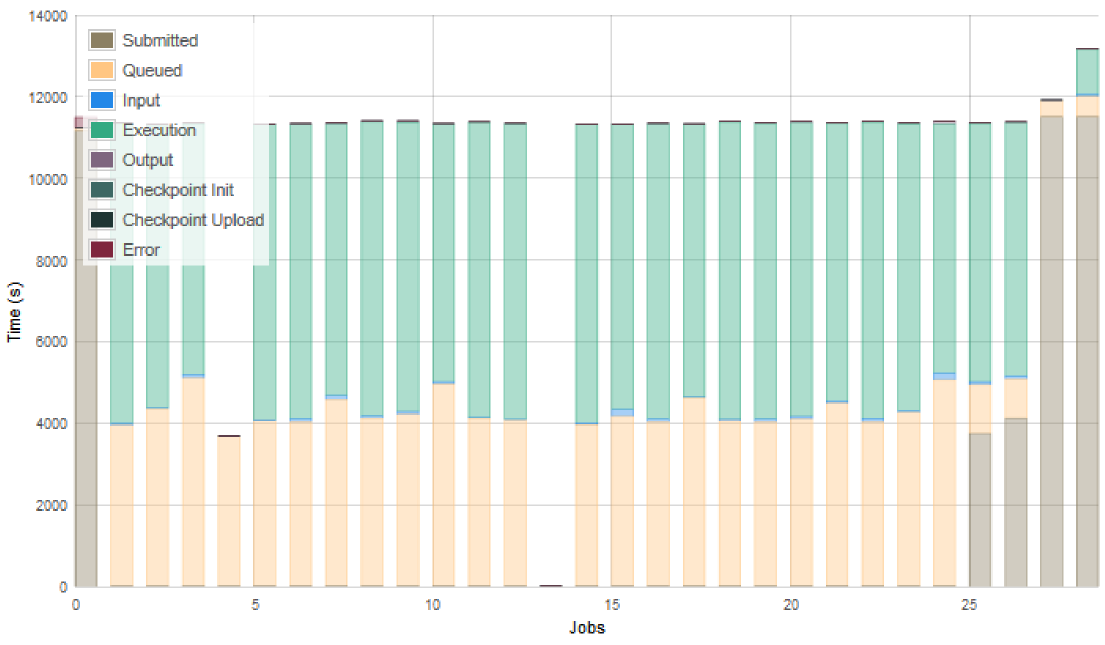


FIGURE 6.7: Job flow for Execution 1. This GATE execution suffers from a long mean waiting time of almost one hour compared to the total makespan of 3.6 hours. Empty bars correspond to failed, canceled or stalled jobs for which we lack information on the running and completion times.

(total CPU time estimated by the user at a few hours). Execution 3 and 4 were launched by *user*2 with 500 computing jobs (total CPU time estimated by the user at more than a few days). Table 6.2 gives more detail on the parameters and performance of the two executions.

The first execution (Figure 6.7) has a cumulated CPU time of 1.9 days and was executed with 25 parallel computing jobs. Its speed-up of 12 (see Table 6.2) is due to a rather long mean waiting time of almost one hour compared to the total makespan of 3.6 hours. The second execution, similar to the first one, benefited from better grid conditions, with a mean waiting time of only two minutes, and achieved a speed up of 21.

The third execution (Figure 6.9) has a cumulated CPU time of 411 days and was executed in 38.5 hours. This execution was impacted by a very important error rate of almost 50% (see Table 6.2). Most of the errors (30% of all jobs) were stalled jobs, i.e. jobs that lost contact with the DIRAC sheduler. This usualy happens when jobs use up the resources (e.g. memory or execution time) allocated by the site to individual job slots. As discussed in Chapter 4, this is frequent for long executions (makespan supperior to 24h), for which checkpointing would be very useful. The rest of the errors were maily data (input and output) transfer errors. Despite the high error rate, the dynamic approach coupled with pilot-jobs managed to produce a result in a relatively good time, with a speed-up of 255.

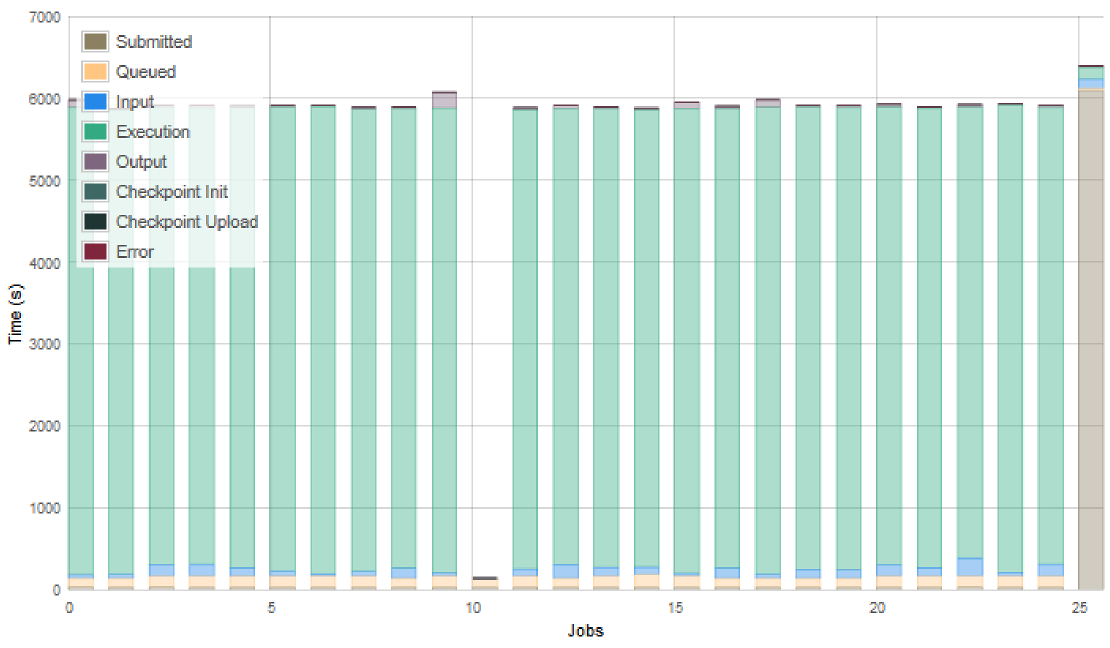


FIGURE 6.8: Job flow for Execution 2. This GATE execution benefits from good grid conditions (mean waiting time of only two minutes). Empty bars correspond to failed, canceled or stalled jobs for which we lack information on the running and completion times.

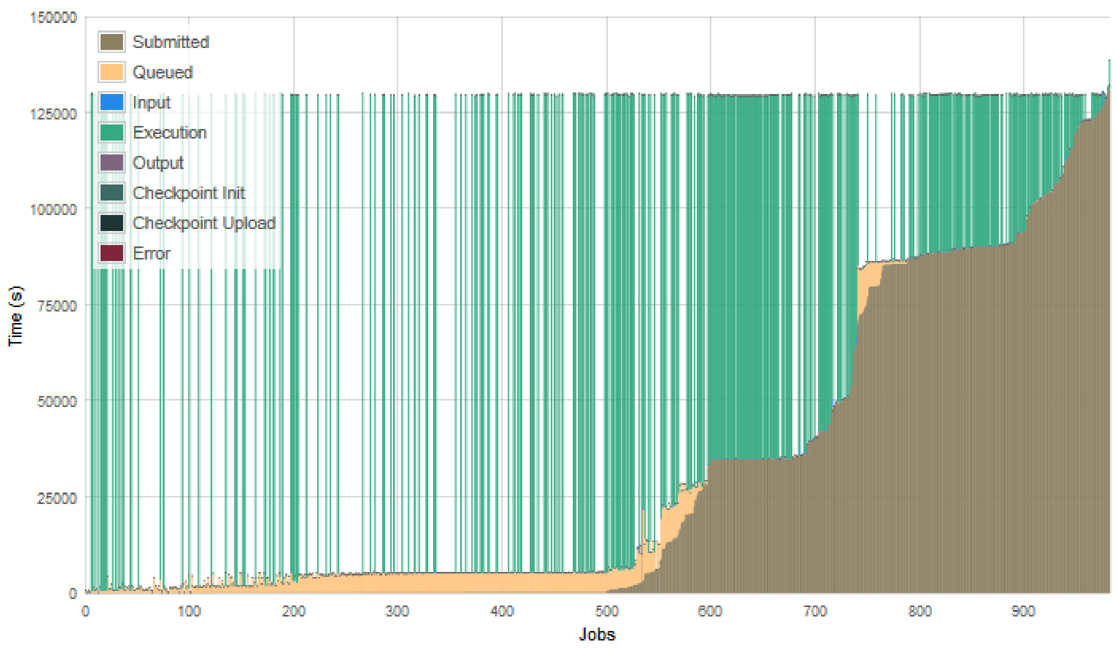


FIGURE 6.9: Job flow for Execution 3. This GATE execution is impacted by a very important error rate of almost 50%. Empty bars correspond to failed, canceled or stalled jobs for which we lack information on the running and completion times.

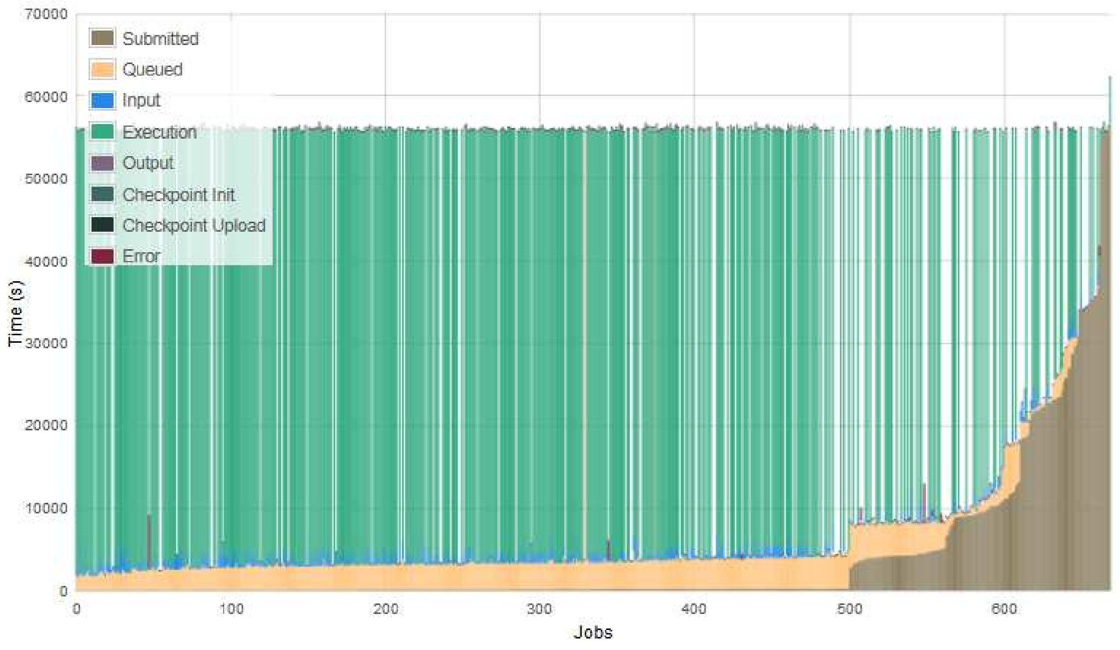


FIGURE 6.10: Job flow for Execution 4. This GATE execution has a remarkable speed-up of 392. Empty bars correspond to failed, canceled or stalled jobs for which we lack information on the running and completion times.

The fourth execution (Figure 6.10) has a cumulated CPU time of 282 days and was executed in 17.3 hours. With an error rate of 25% (mainly input data transfers), this execution managed to reach a remarkable speed-up of 392.

These examples illustrate that the close-to-optimal performance of our dynamic load balancing, showed in controlled conditions is Chapter 3, is still observed in production conditions, on a variety of use-cases and grid conditions.

6.3.3 GATE results obtained with the GATE-Lab

The GATE-Lab is used daily by tens of users executing GATE for different studies. In the following, we will present a few examples of results obtained with the GATE-Lab.

[Gueth et al., 2013] propose a machine learning approach based on Monte-Carlo simulations to create optimized treatment-specific classifiers that detect discrepancies between planned and delivered dose. Simulations were performed with GATE using the GATE-Lab, which reduced execution time from ten days on a single CPU 2.6 GHz Intel Xeon to about 5.2 h (average speed up of 45, including queuing time and the merging of partial results).

Figure 6.11, extracted from [C. Noblet and Delpon, 2013], shows a Monte-Carlo simulation of the irradiation of a C57Bl6 rat using the Xrad225Cx model with 225kV and

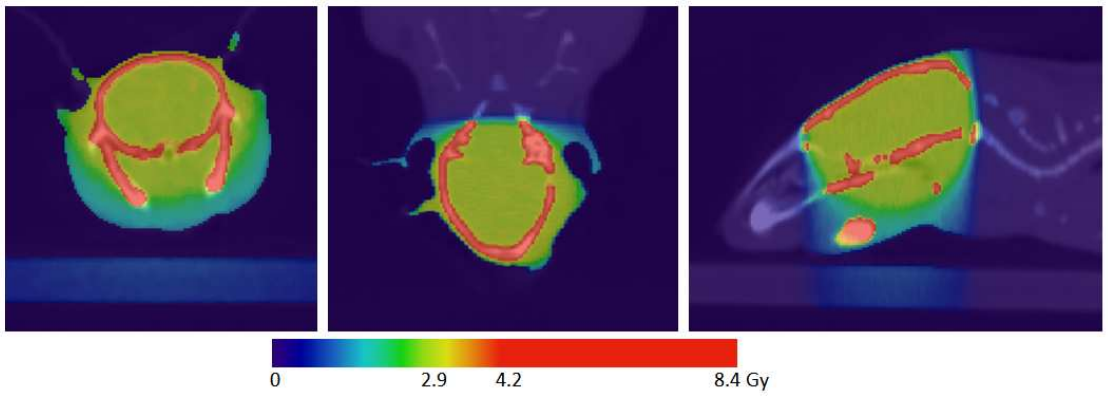


FIGURE 6.11: Monte-Carlo simulation of the irradiation of a C57Bl6 rat using the Xrad225Cx model with 225kV and 13mA. The simulation has a cumulated CPU time of a few months and was completed in one day using the GATE-Lab. Figure extracted from [C. Noblet and Delpon, 2013]

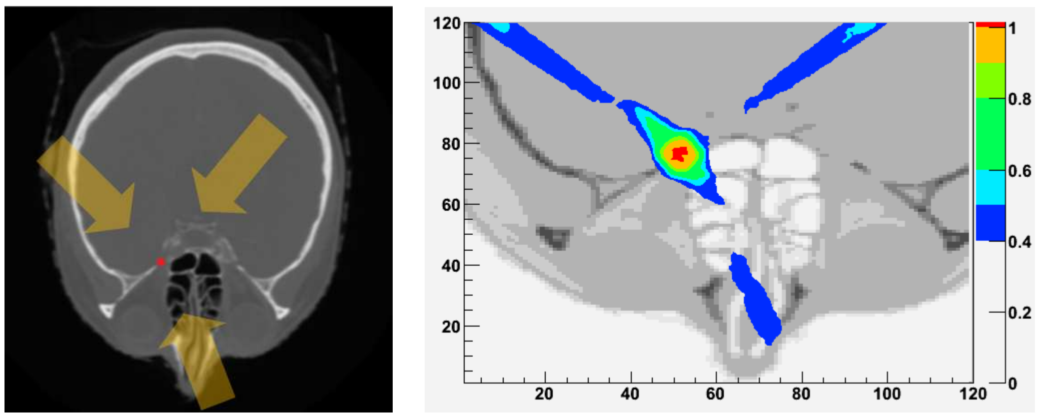


FIGURE 6.12: Treatment planning of ethmoid tumor (in red) using GATE 6.1 with a 6 MeV photon beam, three small fields (1x1 cm2) and 2.5x10E9 primaries. The statistical uncertainty is <1%. The execution corresponds to a total CPU time of approximately 4.5 months completed with the GATE-Lab in less than one day. Credits: Yann Perrot

13mA. The simulation has a cumulated CPU time of a few months and was completed in one day using the GATE-Lab.

The GATE-Lab was also used for the studies presented in Figures 6.12 and 6.13. Figure 6.12 illustrates the treatment planning of ethmoid tumor (in red) using GATE 6.1 with a 6 MeV photon beam, three small fields (1x1 cm2) and 2.5x10E9 primaries. The statistical uncertainty of the result is inferior to 1%. The execution corresponds to a total CPU time of approximately 4.5 months completed with the GATE-Lab in less than one day. Figure 6.13 illustrates a GATE simulation of a beta emitter (iodine 131)in a voxelized mouse phantom derived from micro CT images (backbone in white, lungs in blue, melanoma tumor in yellow).

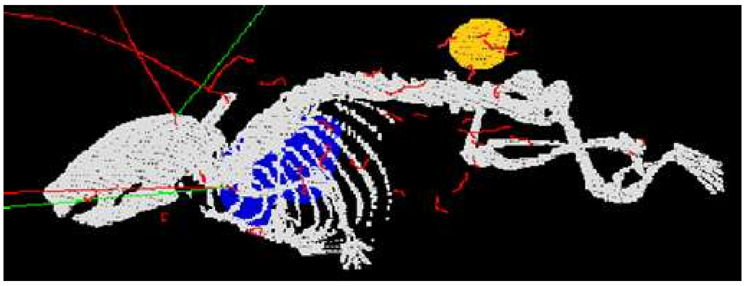


FIGURE 6.13: Simulation with GATE 6.2 of a beta emitter (iodine 131)in a voxelized mouse phantom derived from micro CT images (backbone in white, lungs in blue, melanoma tumor in yellow). Credits: Yann Perrot

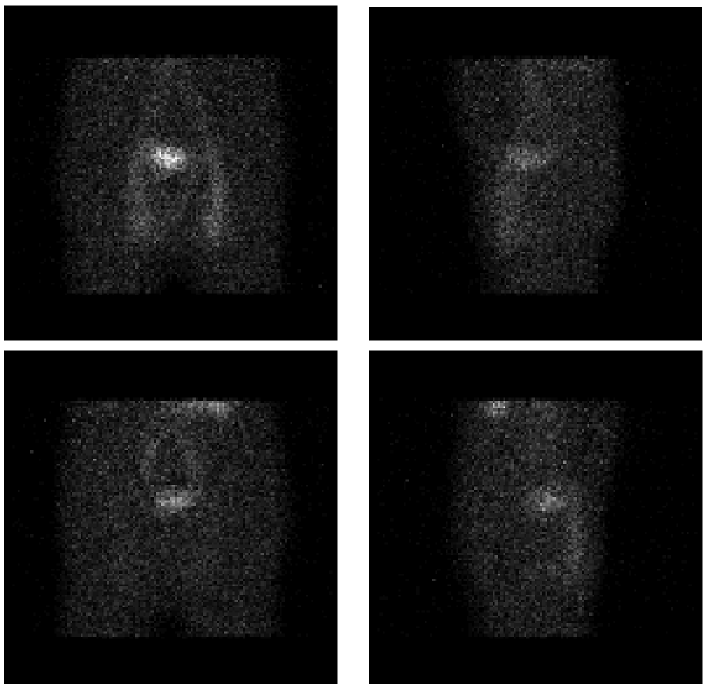


FIGURE 6.14: SPECT (Single Photon Emission Computed Tomography) images for Indium 111 simulated with GATE using a NCAT phantom. Each functional compartment (the blood, liver, urine, kidneys, spleen and remainder body) was simulated independently. A total of 60 projections of 128*128 pixels each were simulated; the four images presented here correspond to the projections at 0, 90, 180 and 270 degrees. Credits: Marie-Paule Garcia

Figure 6.14 illustrates another example of GATE-Lab usage. It shows a SPECT simulation for Indium 111 executed with GATE using a NCAT phantom. Each functional compartment (the blood, liver, urine, kidneys, spleen and remainder body) was simulated independently. A total of 60 projections of 128*128 pixels each were simulated; the four images presented here correspond to the projections at 0, 90, 180 and 270 degrees. The complete simulation corresponds to a total computing time of approximately one year and was computed with the GATE-Lab within a few days.

6.4 Discussion

The GATE-Lab is used more and more intensively, executing computations corresponding to tens of CPU years each month. Nevertheless, its efficiency can still be improved, since the number of failed jobs is still rather high. As discussed previously, most failures are usually due to user mistakes or data transfers.

Among the optimizations proposed in the previous chapters, only the dynamic parallelization is used on a daily basis and has proved to significantly improve performance by reducing the execution makespan. Because of implementation issues, multiple merge and checkpointing are still considered as experimental and are not in production.

The workflow with multiple mergers has actually been publicly available for testing, but its implementation proved not to be sufficiently robust for production usage. While enriched in functionalities, the complete framework proposed in Chapter 4, has become more complex at the workflow description level. New monitoring processes were needed to launch and follow the multiple mergers. In particular, due to their long lifetime, as well as potentially long file transfer and idle times, mergers tend to be killed or become stalled. In this case, the workflow needs to detect the problem and properly handle the partial results on which the merger was working. These functionalities were implemented using multiple Beanshell[4] threads which proved to be unstable for very long workflows. We are currently working on solving these implementation issues and we hope to be able to provide a more robust version before the end of the year.

The use of the checkpointing in production depends, on the one hand, on the multiple mergers and, on the other hand, on the tuning of the checkpointing period. As presented in Chapter 4, the use of checkpointing generates a larger number of partial results which requires the use of multiple mergers. The checkpointing period also needs to be set, ideally depending on the estimated CPU time. Unfortunately, the estimation of the CPU time is very approximate (we may have deltas of a few years of CPU time). Thanks to the results obtained in Chapter 5, Section 5.6 we know that, when choosing a checkpointing period, it is safer to overestimate it than to underestimate it.

6.5 Conclusion

This chapter presented the GATE-Lab, a production system targeting researchers who need to run CPU-intensive GATE executions easily and rapidly, and illustrated usage statistics and results obtained on the VIP/GATE-Lab platform. The GATE-Lab has

4. http://www.beanshell.org

both motivated our contributions and helped validate them beyond controlled experiments. Not only did we integrate our optimizations in the GATE-Lab, but we also made the effort of maintaining and debugging it, and providing user support to whoever needed it.

Results show that the system has been intensively used during the last two years and that its success is partly due to the optimizations proposed in the previous chapters. In particular, the dynamic load-balancing approach has proved to be perfectly adapted to heterogeneous distributed systems such as EGI. Using it in production for the dynamic GATE-Lab executions is not only feasible, but significantly improves performance regardless of the different user executions and grid conditions.

Because of implementation issues, multiple merge and checkpointing are still considered as experimental and are not in production.

Chapter 7

Conclusion and perspectives

In the complex context of heterogeneous distributed systems, our main goal was to propose new strategies for a faster and more reliable execution of Monte-Carlo computations. To achieve this goal, we proposed a new dynamic partitioning algorithm for the computing phase, as well as advanced merging strategies using multiple parallel mergers and checkpointing. We validated our contributions using multiple approaches including experimentation, modeling and simulation. Moreover, we implemented some of them into the VIP/GATE-Lab production platform through which they are used by hundreds of GATE users.

Chapter 3 presented a parallelization approach based on pilots jobs and on a new dynamic load-balancing algorithm. The algorithm consists in a ”do-while” loop with no initial splitting. The master periodically sums up the number of simulated events and sends stop signals to pilots when the total number of requested events is reached. Computing resources are fully exploited until the end of the simulation: resources are released simultaneously up to a negligible time interval. Thus, our load-balancing algorithm can be considered optimal both in terms of resource usage and makespan. Consequently, we consider that the proposed dynamic approach solves the load-balancing problem of particle-tracking Monte-Carlo applications executed in parallel on distributed heterogeneous systems. Nevertheless, this approach presents some limitations since the proposed algorithm: (i) may lead to computing slightly less or more events than initially asked by the user, (ii) renders impossible the exact reproduction of a Monte-Carlo simulation and (iii) is only applicable to Monte-Carlo applications for which all events are strictly equivalent and can be simulated in any order. Beyond coping with these limitations, future work could consist in studying the possibility that the dynamic load-balancing algorithm be used by non-Monte-Carlo simulations.

Chapter 4 completed the optimizations proposed in Chapter 3 with advanced merging strategies using multiple parallel mergers and checkpointing. To evaluate the necessity of the proposed strategies, as well as their performance, three experiments have been conducted on EGI using GATE. 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. The third experiment, with different checkpointing periods, shows that the checkpointing can be used to enable incremental results merging from partial results and to improve reliability without penalizing the makespan. 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 period for the framework with checkpointing. The model is not 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%. Although they proved their efficiency, the multiple merge and checkpointing are not yet in production. While enriched in functionalities, the complete workflow is more complex, more instable and more difficult to debug. Beyond these “implementation” issues, a more fundamental problem still needs to be solved: the proper tuning of the number of mergers and of the checkpointing period.

Chapter 5 proposed an end-to-end simulation of the complete framework introduced in Chapter 4. Based on the SimGrid toolkit, we simulated the deployment of pilot jobs by randomly selecting hosts and matching latencies and failure times to real traces. Job scheduling, application jobs and the main EGI data management services (file catalog and storage element) were also simulated as SimGrid processes. To address performance discrepancies between the real and simulated platforms, the computational cost and file size of the application were calibrated from real traces. The resulting simulator was used to study the influence of the number of mergers in the application workflow, and its checkpointing period. Results show that absolute makespan values of the Monte-Carlo phase are consistent, and that the trends observed in production are correctly reproduced by the simulation. Moreover, parameter-studies results brought deeper insight w.r.t. previous observation on the real system. Indeed, curves obtained for the merging makespan could be used beyond parameter tuning, since they could allow the fitting of mathematical models on the merging makespan.

These results open the door to better and faster evaluation. Nevertheless, it is arguable whether simulation alone can replace other validation methods, such as experimentation in production conditions. Based on our experience, we believe that the two methods should co-exist since they complete each other. Calibration based on extensive production experiments can play an important role in the realism of the simulation as pointed out in Chapter 5. Moreover, without thorough validation against real results, simulation can be misleading. Traces from production experiments can help to calibrate and validate the simulation, which, once validated, can be used to produce further results. In our case, calibration was indispensable since we had no suitable platform model available for EGI. As explained previously, this is mainly due to the lack of detailed information on (i) the configuration (number and performance of processing units, intra-site bandwidth, etc.) of its more than 300 distributed sites and (ii) the inter-site network connections. Gathering the necessary information is particularly challenging because it changes frequently. In order to improve the description of the simulation platform, two approaches could be investigated: (i) calibrating existing platforms, as presented in Chapter 5 or (ii) building an EGI platform model from production traces. Future work could thus consist in investigating the second approach.

Chapter 6 illustrated the applicability and usage of the proposed strategies in production. Counting today some 300 users and more than 150 successful GATE executions per month, the GATE-Lab is a tool that has both motivated these contributions and helped validate them in a production environment. Results show that the system has been intensively used during the last two years and that its success is partly due to the optimizations proposed in the previous chapters. In particular, the dynamic load-balancing approach has proved to be perfectly adapted to heterogeneous distributed systems such as EGI. Using it in production for the dynamic GATE-Lab executions is not only feasible, but significantly improves performance regardless of the different user executions and grid conditions. As a perspective, other Monte-Carlo applications could be integrated into the VIP/GATE-Lab platform and benefit from the dynamic load-balancing approach.

Given the increasing development and popularity of cloud computing technologies, it would be worthwhile to implement the algorithms proposed here on cloud architectures. This would imply that both the Monte-Carlo and the merging tasks execute on cloud computing resources and, therefore, that results are stored on cloud storage resources. The main goal of the strategies presented here was to improve the makespan and the overall reliability (i.e. be sure to provide a final result despite the errors encountered on the platform). When moving into the cloud, the objectives may evolve and optimizations may become more complex since the cost has also to be taken into account. In this context, will the proposed dynamic load balancing algorithm remain optimal w.r.t. cloud costs and other metrics relevant to cloud usage? Merging strategies would also have to be adapted to cloud particularities, where data transfers can be particularly expensive. Nevertheless, they could also benefit from the more localized cloud resources, as well as from the file systems provided on clouds, such as Hadoop.

Chapter 8

Résumé en français

Abstract *Les applications Monte-Carlo sont facilement parallélisables, mais une parallélisation efficace sur des grilles de calcul est difficile à réaliser. Des stratégies avancées d'ordonnancement et de parallélisation sont nécessaires pour faire face aux taux d'erreur élevés et à l'hétérogénéité des ressources sur des architectures distribuées. En outre, la fusion des résultats partiels est également une étape critique. Dans ce contexte, l'objectif principal de notre travail est de proposer de nouvelles stratégies pour une exécution plus rapide et plus fiable des applications Monte-Carlo sur des grilles de calcul. Ces stratégies concernent à la fois le phase de calcul et de fusion des applications Monte-Carlo et visent à être utilisées en production.*

Dans cette thèse, nous introduisons une approche de parallélisation basée sur l'emploi des tâches pilotes et sur un nouvel algorithme de partitionnement dynamique. Les résultats obtenus en production sur l'infrastructure de grille européenne (EGI) en utilisant l'application GATE montrent que l'utilisation des tâches pilotes apporte une forte amélioration par rapport au système d'ordonnancement classique et que l'algorithme de partitionnement dynamique proposé résout le problème d'équilibrage de charge des applications Monte-Carlo sur des systèmes distribués hétérogènes. Puisque toutes les tâches finissent presque simultanément, notre méthode peut être considérée comme optimale à la fois en termes d'utilisation des ressources et de temps nécessaire pour obtenir le résultat final (makespan).

Nous proposons également des stratégies de fusion avancées avec plusieurs tâches de fusion. Une strategie utilisant des sauvegardes intermédiaires de résultat (checkpointing) est utilisée pour permettre la fusion incrémentale à partir des résultats partiels et pour améliorer la fiabilité. Un modèle est proposé pour analyser le comportement de la plateforme complète et aider à régler ses paramètres. Les résultats expérimentaux montrent que le *modèle correspond à la réalité avec une erreur relative de 10% maximum, que l'utilisation de plusieurs tâches de fusion parallèles réduit le temps d'exécution total de 40% en moyenne, que la strategie utilisant des sauvegardes intermédiaires permet la réalisation de très longues simulations sans pénaliser le makespan.*

Pour évaluer notre équilibrage de charge et les stratégies de fusion, nous mettons en oeuvre une simulation de bout-en-bout de la plateforme décrite ci-dessus. La simulation est réalisée en utilisant l'environnement de simulation SimGrid. Les makespan réels et simulés sont cohérents, et les conclusions tirées en production sur l'influence des paramètres tels que la fréquence des sauvegardes intermédiaires et le nombre de tâches de fusion sont également valables en simulation. La simulation ouvre ainsi la porte à des études paramétriques plus approfondies.

8.1 Introduction

Les simulations Monte-Carlo sont utilisées dans plusieurs domaines scientifiques pour produire des résultats réalistes à partir d'un échantillonnage répété d'événements générés par des algorithmes pseudo-aléatoires. La méthode de Monte-Carlo a été définie par [Halton, 1970] comme “représentant la solution d'un problème en tant que paramètre d'une population hypothétique, et en utilisant une séquence aléatoire de nombres pour construire un échantillon de la population, à partir duquel on peut obtenir des estimations statistiques du paramètre recherché”. La Figure 8.1 illustre un exemple classique de la façon dont la méthode Monte-Carlo peut être appliquée pour estimer la valeur de π. En considérant un cercle inscrit dans un carré unité, la méthode consiste à (i) disperser de manière uniforme des objets de taille uniforme sur la surface du carré et (ii) à compter le nombre d'objets à l'intérieur du cercle et le nombre total d'objets à l'intérieur du carré. Le rapport des deux valeurs est une estimation du rapport des deux superficies, à savoir $\pi/4$.

Dans le domaine de l'imagerie médicale et de la radiothérapie, GATE [Allison et al., 2006] est un simulateur Monte-Carlo de transport de particules développé par la collaboration internationalle OpenGATE[1]. La simulation Monte-Carlo de transport de particules consiste en un suivi stochastique successif à travers la matière d'un grand nombre de particules individuelles. Chaque particule a un ensemble initial de propriétés (type, emplacement, direction, énergie, etc) et son interaction avec la matière est déterminée en fonction des probabilités d'interaction réalistes et des distributions angulaires. GATE permet actuellement de réaliser des simulations de tomographie (CT) , tomographie par émission (Tomographie par Emission de Positons - TEP - et Tomographie à

1. http://www.opengatecollaboration.org

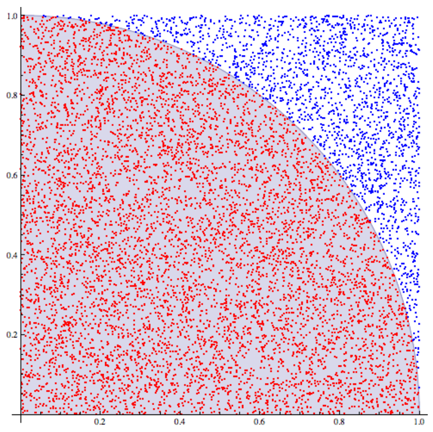


FIGURE 8.1: Utilisation de la méthode Monte-Carlo pour approximer la valeur de π. Avec 30000 points repartis de manière aléatoire, l'erreur d'estimation est inférieure à 0.07%. Crédits : Wikipedia

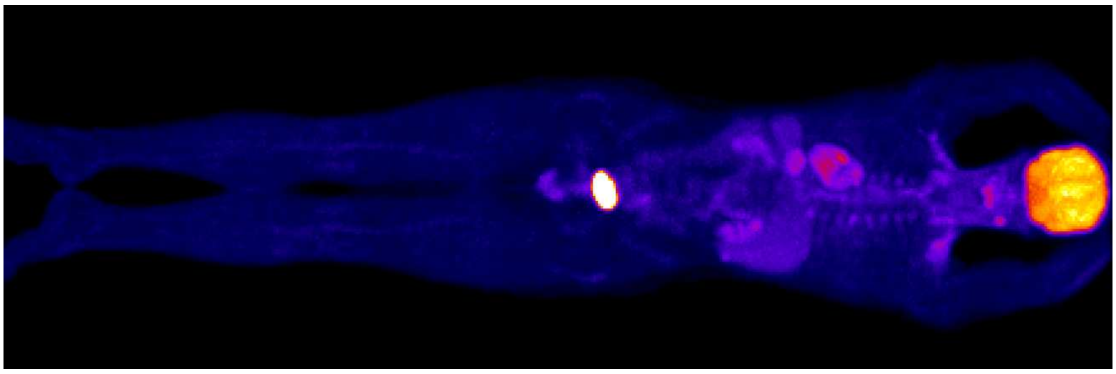


FIGURE 8.2: Scan 3D d'une TEP F18-FDG corps entier simulé avec GATE et représentant 4000 heures CPU (5,3 mois). Crédits : IMNC-IN2P3 (CNRS UMR 8165).

Emission Mono-Photonique - TEMP) et radiothérapie. Utilisé par une grande communauté internationale, GATE joue désormais un rôle clé dans la conception de nouveaux appareils d'imagerie médicale, dans l'optimisation des protocoles d'acquisition et dans le développement et l'évaluation d'algorithmes de reconstruction d'image et des techniques de correction. Il peut également être utilisé pour le calcul de dose dans des expériences de radiothérapie.

Pour produire des résultats précis, les simulations Monte-Carlo nécessitent un grand nombre d'événements statistiquement indépendants, ce qui a un fort impact sur le temps de calcul de la simulation. À titre d'exemple, la figure 8.2 illustre un scan 3D d'une Tomographie par Emission de Positons (TEP) F18-FDG corps entier simulée avec GATE et représentant 4000 heures CPU (5,3 mois). Pour faire face à un tel défi de calcul, les simulations Monte-Carlo sont couramment parallélisées en utilisant un patron de divison et fusion (split and merge pattern). Elles peuvent ainsi exploiter d'importantes quantités de ressources distribuées disponibles dans le monde entier.

Le calcul distribué est une méthode de calcul largement utilisée, où les différentes parties d'un programme sont traitées simultanément sur deux ou plusieurs ordinateurs qui communiquent entre eux sur un réseau. Le calcul distribué est apparu suite à la création des grappes de calcul (clusters), qui ont ensuite été reliées en grilles de calcul. Aujourd'hui, le calcul distribué continue d'évoluer avec la technologie de nuage (cloud). Alors que les noeuds d'une grappe de calcul sont reliés par un réseau local (LAN), les grilles et les nuages sont répartis géographiquement, couvrant généralement plusieurs domaines administratifs. Une comparaison détaillée entre ces technologies est disponible dans [Sadashiv and Kumar, 2011]. Les technologies de grille sont principalement utilisées par les systèmes de calcul à haut débit (High Throughput Computing - HTC), qui se concentrent sur l'exécution d'un grand nombre de tâches indépendantes, par opposition aux systèmes de calcul haute performance (High Performance Computing - HPC), qui se concentrent sur l'exécution de tâches fortement couplées. Dans cette thèse, nous nous référerons essentiellement aux grilles de calcul et, par conséquent, à des systèmes HTC.

Diverses stratégies ont été proposées pour accélérer l'exécution des simulations Monte-Carlo sur plateformes distribuées [Maigne et al., 2004, Stokes-Ress et al., 2009]. Néanmoins, elles utilisent un découpage statique des simulations et ne sont pas bien adaptées aux grilles, où les ressources sont hétérogènes et peu fiables. En outre, elles se concentrent principalement sur la phase de calcul de la simulation alors que la fusion des résultats partiels est également une étape critique. La fusion des résultats partiels est une opération très sensible au nombre de résultats partiels produits, à la disponibilité des ressources de stockage, ainsi qu'au débit et à la latence du réseau. Sur des infrastructures de taille mondiale, les résultats partiels sont généralement distribués géographiquement proche de leur site de production, ce qui aggrave le coût des transferts de données. Dans certains cas, le temps de fusion peut même devenir comparable au makespan de simulation.

Les taux d'erreur élevés ont de fortes conséquences sur les performances des applications. Des taux d'erreur de plus de 10% sont couramment observés sur les infrastructures de production comme EGI [Jacq et al., 2008]. En conséquence, la tolérance aux pannes [Garg and Singh, 2011] devient essentielle pour le calcul sur grille. Les techniques le plus couramment utilisées pour fournir la tolérance aux pannes sont l'utilisation de tâches pilotes, la sauvegarde intermédiaire des résultats (checkpointing) et la réplication des tâches. Les tâches pilotes soumettent des tâches génériques qui récupèrent et exécutent les tâches de l'utilisateur dès qu'ils commencent à s'exécuter sur les ressources disponibles. Ainsi, la tolérance aux pannes se manifeste à deux niveaux : tout d'abord, les tâches pilotes vérifient leur environnement d'exécution avant de récupérer les tâches des utilisateurs, ce qui réduit le taux d'erreur des tâches de l'utilisateur ; d'autre part, afin d'éviter les erreurs récurrentes, les pilotes en erreur sont retirés et les tâches échouées sont éxécutées par un autre pilote. De plus, les erreurs qui surviennent avant que les pilotes n'atteignent une ressource de calcul n'impactent pas les tâches. La sauvegarde intermédiaire de résultats est souvent utilisée pour améliorer la fiabilité de l'application, mais elle doit être bien réglée pour limiter le surcoût. La sauvegarde intermédiaire de résultats est également intéressante à étudier car elle permet la production et la fusion incrémentale des résultats. La réplication des tâches consiste à distribuer plusieures répliques d'une même tâche et à utiliser le résultat de la réplique qui finit en premier. Cette stratégie peut atteindre une très bonne tolérance aux pannes, mais les tâches redondantes gaspillent des cycles de calcul qui pourraient être utilisés pour d'autres applications.

La performance des stratégies, telles que des algorithmes d'ordonnancement, utilisées sur les infrastructures de calcul distribuées est difficile à évaluer en production à cause des conditions d'exécution variables d'une expérience à l'autre. Les méthodes d'évaluation peuvent combiner plusieurs approches, telles que la modélisation, l'expérimentation et la simulation. La modélisation peut aider à comprendre le comportement des applications distribuées et à régler les paramètres comme la période entre deux sauvegardes intermédiaires. Néanmoins, la modélisation représente un défi pour les applications s'exécutant en production parce que la plupart des paramètres, comme par exemple la charge de fond de l'infrastructure ou les caractéristiques des ressources impliquées dans l'exécution, ne sont pas connus avant la fin de l'expérience. Les modèles utilisés dans la production doivent être en mesure de faire face à un tel manque d'information, en mettant l'accent sur les paramètres qui sont mesurables par les applications. D'un autre côté, les expériences en production assurent des conditions et des résultats réalistes, mais sont lourdes et difficiles à reproduire. La reproduction des expériences en simulation devient intéressante, car elle permet d'expérimenter rapidement et d'assurer des conditions contrôlées.

Dans ce contexte, le principal défi adressé par cette thèse consiste à trouver de nouvelles stratégies pour une exécution plus rapide et plus fiable des calculs Monte-Carlo sur des infrastructures hétérogènes distribuées. Ces stratégies concernent à la fois les étapes de fusion et de calcul des applications Monte-Carlo et visent à être utilisées en production. De plus, elles seront validées par des approches multiples, y compris l'expérimentation, la modélisation et la simulation.

Le reste du chapitre est organisé comme suit : la Section 8.2 introduit un algorithme d'équilibrage de charge dynamique, la Section 8.3 présente des stratégies avancées de fusion de résultats et la Section 8.4 décrit la simulation de la plateforme proposée en utilisant SimGrid. Le chapitre sera cloturé par les conclusions et les perspectives de cette thèse. Notons que les trois sections mentionnées ci-dessous correspondent respectivement aux chapitres 3, 4 et 5. L'état de l'art et l'illustration des résultats obtenus en production par les utilisateurs de la plateforme VIP, présentés respectivement dans les chapitres 2 et 6, n'ont pas été repris dans ce résumé en français.

8.2 Algorithme d'équilibrage de charge dynamique

Dans des environnements hétérogènes non fiables comme EGI, les approches statiques de partitionnement mènent à un mauvais équilibrage de charge. Pour illustrer ce point, la Figure 8.3 presente le flot d'exécution d'une simulation GATE divisée de manière statique en 75 tâches et exécutée sur EGI. L'équilibrage de charge est clairement sous-optimal : au cours de la dernière heure de simulation, il y a moins de 6 tâches s'exécutant en parallèle, ce qui sous-exploite de manière évidente les ressources disponibles. L'hétérogénéité a un fort impact conduisant à des tâches très longues (par exemple, la tâche 16) et d'autres très courtes (par exemple la tâche 8). En outre, les erreurs conduisent à des resoumissions qui pénalisent encore plus l'exécution.

Nous sommes donc à la recherche d'une stratégie d'équilibrage de charge dynamique qui permettrait de répartir la charge de calcul sur les ressources disponibles en fonction de leur performance. Compte tenu de l'échelle de l'infrastructure de grille visée, les communications entre les tâches, entre les tâches et le maître et entre les tâches et les éléments de stockage doivent être évitées autant que possible. En outre, la réplication des tâches doit être utilisée avec modération sur des infrastructures distribuées telles que EGI pour limiter le surcoût vis-à-vis du reste des utilisateurs.

Une approche classique de parallélisation statique pour les simulations Monte-Carlo de transport de particules consiste à distribuer le nombre total d'événements N en T tâches, chaque tâche recevant une fraction N/T du nombre total d'événements. Dans le cas où des tâches pilotes sont utilisées, le maître distribue des tâches aux pilotes disponibles jusqu'à ce que toutes les tâches soient terminées avec succès. A la fin de chaque tâche, les pilotes chargent leur résultat et se voient attribuer une nouvelle tâche si disponible.

Les tâches pilotes peuvent ainsi équilibrer dynamiquement le nombre d'événements si on leur assigne plusieures petites tâches l'une après l'autre. Une granularité très fine des tâches (par exemple un événement par tâche) est équivalente à un partitionnement dynamique (chaque pilote exécute des simulations d'un événement jusqu'à ce que la simulation soit complète) mais présente un surcoût important (communication avec le maître, transferts des fichiers entrée/sortie, démarrage de l'application, etc.) Inversement, une granularité grossière des tâches (grand nombre d'événements par tâche, par conséquent petit nombre de tâches par rapport aux ressources disponibles) réduit le surcoût mais devient équivalente à une approche statique, conduisant à un mauvais équilibrage de charge dans des environnements hétérogènes non fiables comme EGI (voir Figure 8.3).

Lorsque des tâches pilotes sont utilisées, le maître réaffecte automatiquement les tâches ayant échoué à d'autres pilotes en cours d'exécution. Les erreurs sont ainsi traitées plus

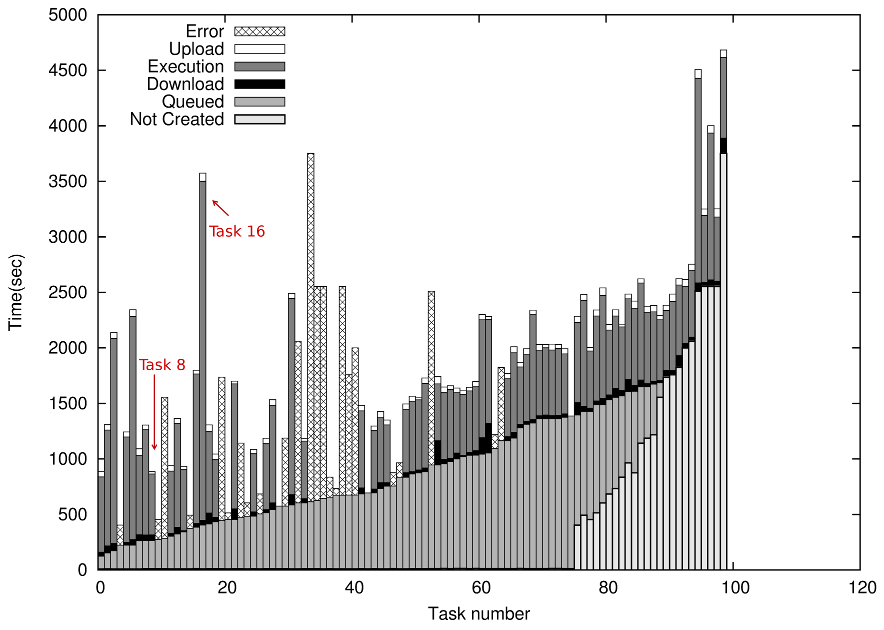


Figure 8.3: Exemple de flot d'exécution d'une simulation GATE divisée de manière statique en 75 tâches et exécutée sur EGI. L'équilibrage de charge est clairement sous-optimal : au cours de la dernière heure de simulation, il y a maximum 6 tâches en parallèle, ce qui sous-exploite de manière évidente les ressources disponibles.

rapidement qu'avec un système d'ordonnancement classique où les tâches qui ont échoué sont mises en attente jusqu'à ce que de nouvelles ressources soient disponibles.

8.2.1 Algorithme dynamique

L'équilibrage de charge dynamique proposé ici consiste en une boucle "tant que" sans découpage initial. Chaque tâche d'une simulation est créée avec le nombre total d'événements et continue à s'exécuter jusqu'à ce que le nombre souhaité d'événements soit atteint avec la contribution de toutes les tâches. Par conséquent, chaque tâche peut simuler l'ensemble de la simulation si les autres tâches échouent ou ne démarrent pas. Le nombre total d'événements simulés est obtenu par la somme de tous les événements simulés par des tâches indépendantes. Ainsi, chaque ressource contribue à l'ensemble de la simulation jusqu'à ce qu'elle se termine.

Les algorithmes 3 et 4 présentent le pseudo-code du maître et des pilotes. Le maître somme régulièrement le nombre d'événements simulés et envoie des signaux d'arrêt aux pilotes en cas de besoin. Chaque pilote exécute une seule tâche, en commençant dès que le pilote arrive sur un noeud de grille et s'arrêtant à la fin de la simulation. Des communications très ponctuelles entre les tâches et le maître sont nécessaires pour mettre

Algorithme 3 Algorithme du maître pour l'équilibrage de charge dynamique des simulations Monte-Carlo

```
N=nombre total d'événements à simuler
n=0
Tant que n<N faire
    n = nombre d'événements simulés par les tâches en exécution ou terminées avec
    succès
Fin tant que
Envoyer signal d'arrêt à toutes les tâches
Annuler les tâches en attente
```

Algorithme 4 Algorithme des pilotes pour l'équilibrage de charge dynamique des simulations Monte-Carlo

```
Télécharger les données en entrée
N=nombre total d'événements à simuler
n=0, dernièreMiseàJour=0, delaiMiseàJour=5min
Tant que signal d'arrêt non reçu ET n<N faire
    Simuler l'événement suivant
    n++
    Si (getTime() - dernièreMiseàJour) >delaiMiseàJour alors
        Envoyer n au maître
        dernièreMiseàJour = getTime()
    Fin si
Fin tant que
Charger le résultat
```

à jour le nombre actuel d'évènements calculés, ainsi qu'à la fin de la simulation lorsque le maître envoie des signaux d'arrêt.

Les erreurs sont efficacement traitées par cet algorithme. Lorsqu'une tâche échoue, ceux qui restent continuent à s'exécuter jusqu'à ce que tous les événements aient été simulés. Aucune resoumission de tâche n'est donc nécessaire.

8.2.2 Expériences et résultats

Le but des expériences présentées dans cette section est de comparer la parallélisation statique classique avec notre parallélisation dynamique, les deux approches utilisant des tâches pilotes DIANE [Mościcki, 2003]. Nous appellerons le scénario statique DS et le scénario dynamique DD.

Chaque expérience consiste à exécuter une simulation GATE avec 450000 événements, représentant 16 heures[2] de calcul. Les deux approches sont testées avec 25, 50 et 75 pilotes et un nombre égal des tâches (c'est à dire respectivement 25, 50 et 75 tâches). Pour l'approche statique, les simulations sont divisées en tâches calculant respectivement

2. Durée moyenne lorsque la simulation est exécutée sur des ressources EGI

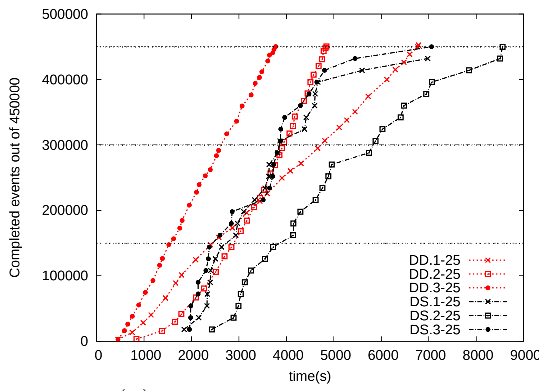
18000, 9000 et 6000 événements. Pour l'approche dynamique, toutes les tâches sont assignées le nombre total de 450000 événements. Chacun des 2 scenarii se compose de 3 expériences différentes répétées 3 fois et soumises simultanément.

Les pilotes en erreur sont retirés pour éviter de nouvelles défaillances et ne sont pas resoumis. Pour l'implémentation statique, les tâches qui ont échoué sont réaffectées aux pilotes enregistrés. Pour l'implémentation dynamique, la ressoumission des tâches n'est pas nécessaire puisque les pilotes qui n'échouent pas calculent jusqu'à la fin de la simulation et puisque nous soumetons un nombre de pilotes égal au nombre de tâches : les tâches resoumises resteraient en file d'attente jusqu'à la fin de la simulation.

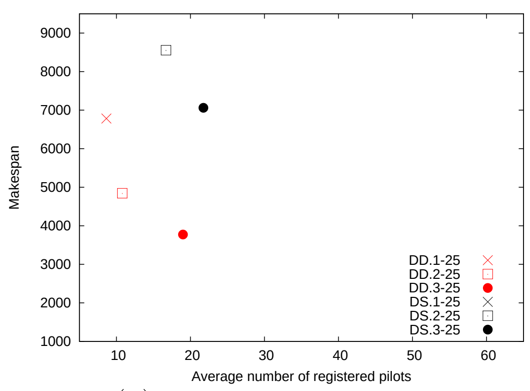
La Figure 8.4 compare les performances des implémentations dynamique et statique. Nous remarquons que la parallélisation dynamique apporte une amélioration significative des performances, le makespan étant en moyenne 2 fois plus petit. En effet, il y a un gain de temps important au niveau des dernières tâches. Grâce à l'absence des ressoumissions et à une meilleure exploitation des ressources, le débit de calcul (événements/s) reste constant jusqu'à la fin de la simulation pour l'approche dynamique. Dans la figure 8.4 (a), la mauvaise performance de la première répétition du scénario dynamique avec 25 pilotes (DD.1-25) est dûe au faible nombre de pilotes enregistrés.

La Figure 8.5 illustre le flot d'exécution des tâches obtenu avec l'algorithme dynamique appliqué à une simulation GATE exécutée sur EGI avec 75 pilotes. L'équilibrage de charge dynamique est nettement supérieur à celui obtenu avec un partitionnement statique (voir Figure 8.3). Les erreurs sont compensées sans resoumissions. L'équilibrage de charge dynamique réussit à résoudre les problèmes (erreurs, des ressources de rechange ou lente) de l'approche statique.

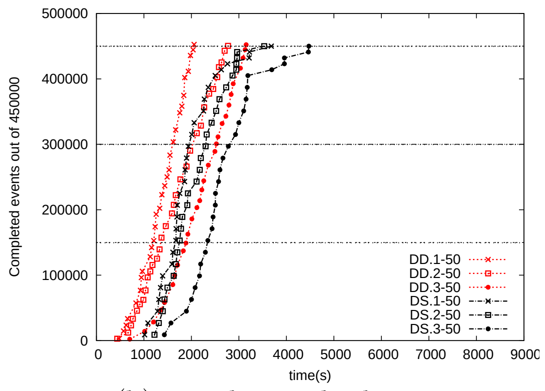
Toutes les tâches finissent presque simultanément, ce qui rend notre méthode optimale tant en termes d'utilisation des ressources et de makespan. Les résultats montrent que l'algorithme proposé apporte des accélerations de l'ordre de deux comparé à une parallélisation statique. Par conséquent, nous considérons que l'approche dynamique proposée résout le problème d'équilibrage de charge des applications Monte-Carlo exécutées en parallèle sur des systèmes distribués et hétérogènes. Néanmoins, la phase de calcul doit être suivie d'une phase de fusion permettant de produire le résultat final. La section suivante propose de nouvelles solutions pour améliorer encore plus la performance globale des applications Monte-Carlo en prenant en compte l'optimisation de l'étape de fusion.



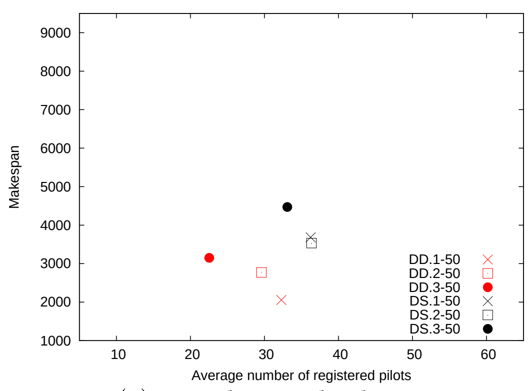
(a) 25 submitted pilots



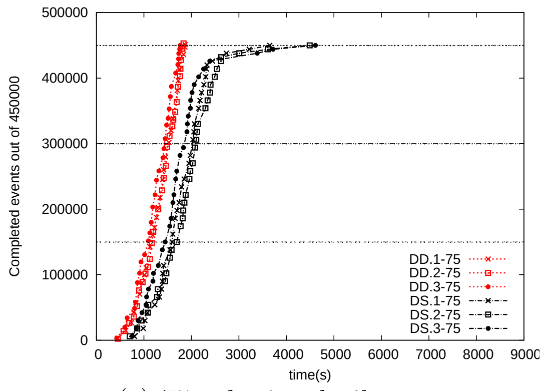
(d) 25 submitted pilots



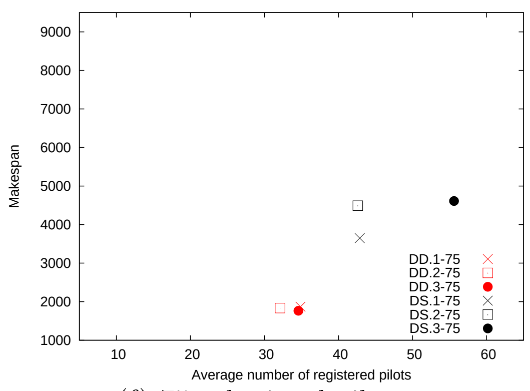
(b) 50 submitted pilots



(e) 50 submitted pilots



(c) 75 submitted pilots



(f) 75 submitted pilots

Evénements simulés au cours du temps

Makespan en fonction des pilotes enregistrés

FIGURE 8.4: Performance des approches dynamique et statique. Les deux scenarii sont dessinés avec un style de ligne différent ; les expériences menées en parallèle sont tracées avec les mêmes symboles (étoile, cercle ou carré). Sur chaque figure, trois lignes horizontales sont tracées à 33%, 66% et 100% des événements simulés. La parallélisation dynamique apporte une amélioration significative des performances car le makespan est considérablement (jusqu'à deux fois) plus petit.

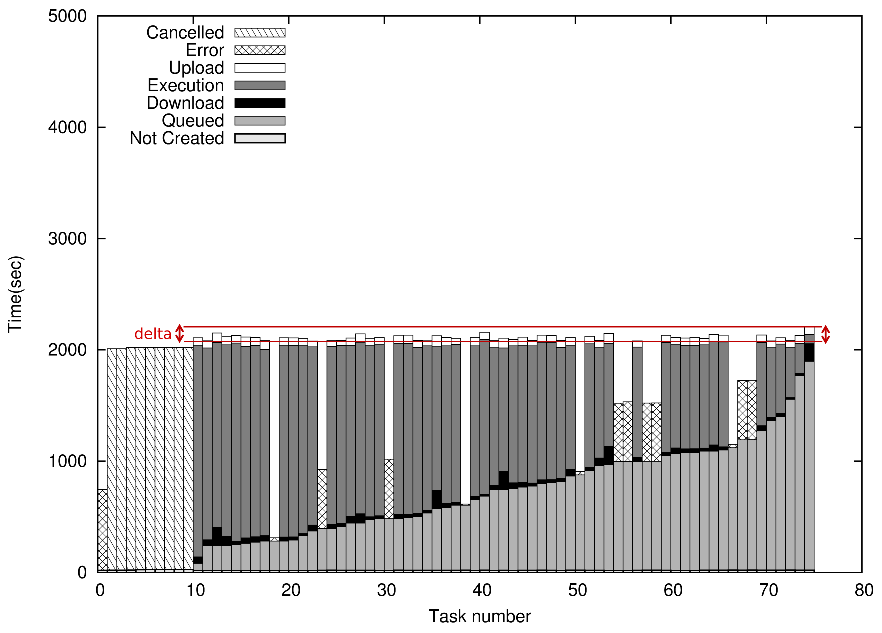


FIGURE 8.5: Example de flot d'exécution des tâches obtenu avec l'algorithme dynamique appliqué à une simulation GATE exécutée sur EGI avec 75 pilotes. Les ressources disponibles sont toutes exploitées jusqu'à la fin de la simulation. Les erreurs sont compensées sans resoumission. L'équilibrage de charge dynamique est nettement supérieur à celui obtenu avec un partitionnement statique (voir Figure 8.3).

8.3 Stratégies de fusion avancées

8.3.1 Plateforme proposée

La phase de fusion d'une simulation Monte-Carlo parallélisée sur une grille consiste à télécharger et fusionner tous les résultats partiels. Sachant que les résultats partiels sont distribués géographiquement sur plusieurs sites, leur temps de transfert est très sensible à la disponibilité des ressources de stockage, ainsi qu'au débit et à la latence du réseau [Ma et al., , Li et al., 2010]. Le problème est aggravé par la production quasi simultanée des résultats partiels à la fin de la phase de calcul, car l'équilibrage dynamique de charge utilise toutes les ressources jusqu'à la fin de la simulation.

Pour illustrer cela, la figure 8.6 montre le flot d'exécution d'une simulation GATE. Les tâches de calcul finissent presqu'en même temps, à peu près au même moment que la tâche de fusion commence à s'exécuter. Dans cet exemple, le temps de fusion représente environ 20% du makespan total, mais dans d'autres cas, dû aux délais de transfert extrêmement longs de quelques résultats partiels, il peut même devenir comparable au makespan de la simulation. Notre objectif est donc de produire et livrer le résultat final le plus tôt possible après la fin de la phase de calcul.

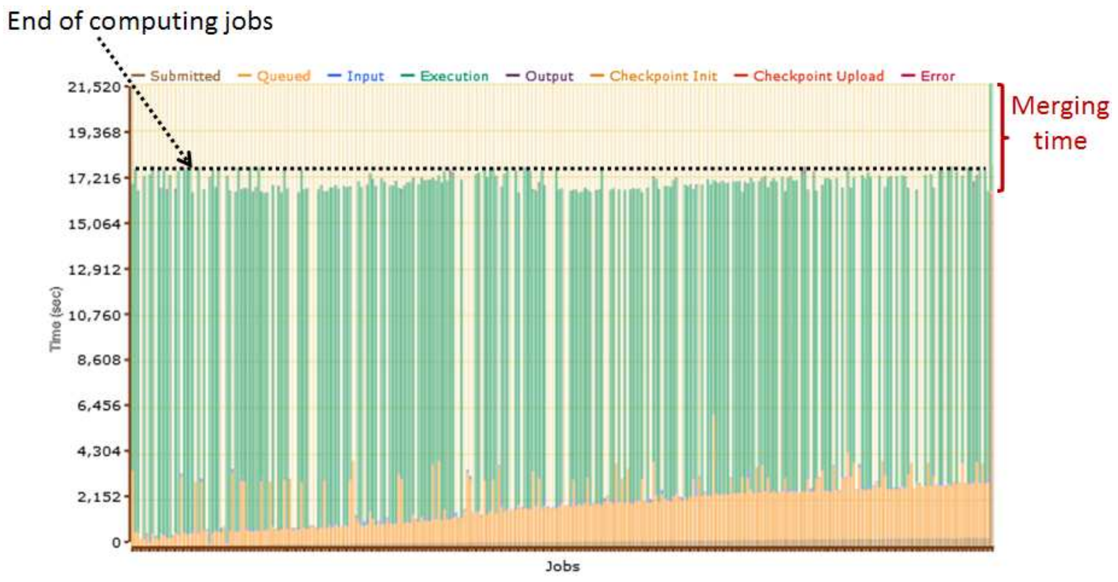


FIGURE 8.6: Flot d'exécution d'une simulation GATE sans sauvegarde intermédiaire des résultats et avec une seule tâche de fusion. Les tâches de calcul finissent presqu'en même temps, à peu près au même moment que le début d'exécution de la tâche de fusion. Les barres vides correspondent aux tâches pour lesquelles nous manquons d'information sur leurs temps d'exécution (cela peut arriver dans le cas des tâches en erreur ou annulées).

La sauvegarde intermédiaire des résultats est souvent utilisée pour améliorer la fiabilité de l'application. Dans notre cas, elle est également intéressante pour permettre une fusion incrémentale des résultats partiels pendant la phase de calcul. Néanmoins, la sauvegarde intermédiaire doit être bien réglée afin de limiter le surcoût. Une période trop courte serait trop coûteuse en raison du très grand nombre de résultats partiels produits, tandis qu'une trop longue période réduirait le bénéfice de la fusion progressive.

Dans les applications Monte-Carlo considérées ici, le processus de fusion est commutatif et associatif. Nous pouvons donc utiliser plusieurs tâches de fusion parallèles afin de réduire le makespan. Cette stratégie distribue non seulement l'opération de fusion elle-même, mais également les transferts des résultats partiels.

La Figure 8.7 présente l'algorithme mis en place dans la section précédente et étendu avec des tâches de fusion multiples. Le maître génère plusieures tâches de simulation Monte-Carlo parallèles, chacune avec le nombre total n d'événements à simuler. Chaque tâche transmet régulièrement son nombre d'événements simulés au maître. À ce stade, les événements se trouvent sur le disque local et ne sont pas disponibles pour la fusion. Les tâches de simulation vérifient ensuite si le maître a envoyé le signal d'arrêt (si le nombre total d'événements simulés à partir de toutes les tâches de simulation est atteint). Cette partie correspond à l'algorithme d'équilibrage de charge dynamique présentée dans la section précédente. Lorsque la condition d'arrêt est atteinte, le maître lance plusieures

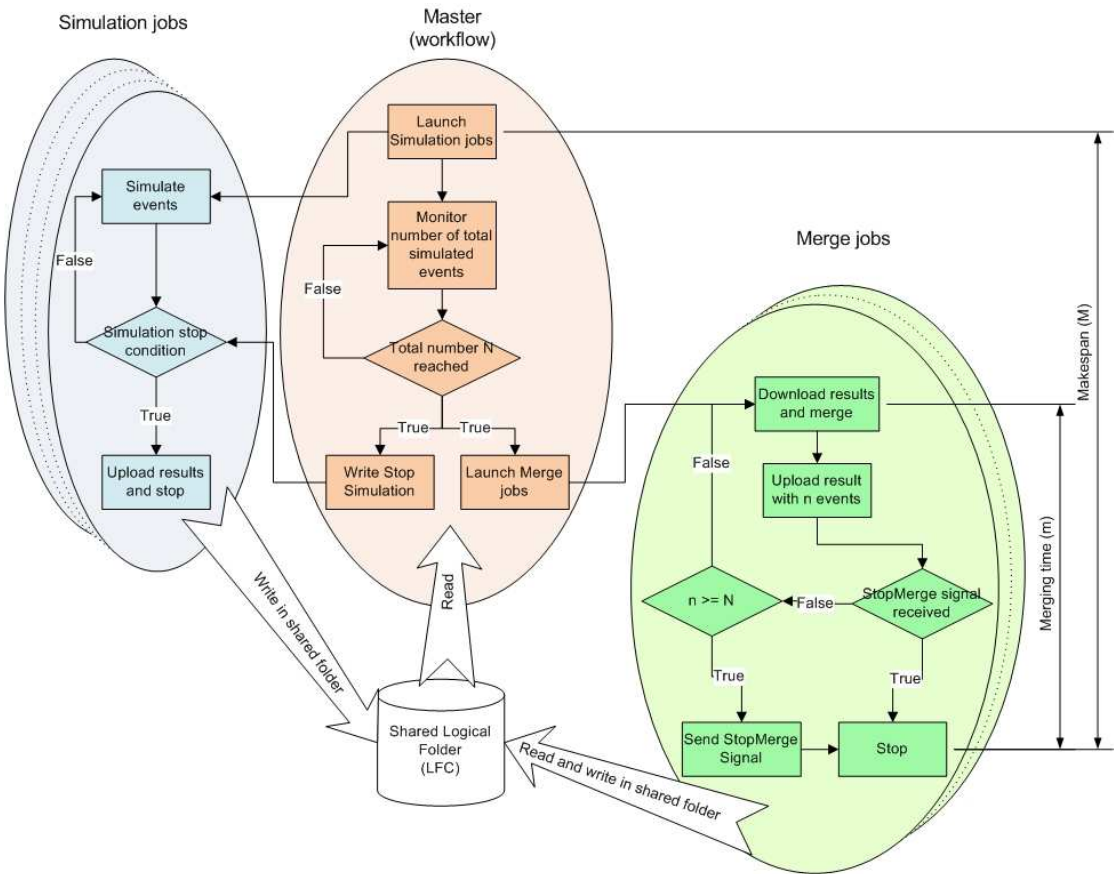


FIGURE 8.7: Algorithme sans sauvegarde intermédiaire des résultats. Les tâches de simulation sont représentées en bleu, les tâches de fusion en vert et le maître en orange. Plusieures tâches de simulation et de fusion sont exécutées en parallèle, mais, pour des raisons de lisibilité, une seule tâche de simulation et une tâche de fusion sont représentées. Les tâches de simulation chargent leurs résultats une fois à la fin de leur vie. La condition `SimulationStop` donnée par le maître déclenche : i) le chargement des résultats de simulation et ii) le lancement des tâches de fusion. Le signal `StopMerge` est envoyé par la première tâche de fusion ayant fusionné le nombre requis d'événements.

tâches de fusion et les tâches de simulation chargent leurs résultats partiels dans un dossier logique partagée (LFC). Le contenu de ce dossier peut être stocké physiquement sur plusieurs éléments de stockage répartis géographiquement. Tant qu'il n'y a pas de signal `StopMerge`, les tâches de fusion sélectionnent, téléchargent et fusionnent des lots de résultats partiels. Leur résultat est ensuite envoyé vers le dossier partagé pour une nouvelle fusion ultérieure. Si une tâche de fusion produit un résultat contenant le nombre total d'événements, elle envoie également le signal `StopMerge`.

Afin de permettre une fusion incrémentale des résultats partiels, nous utilisons des sauvegardes intermédiaires des résultats. La plateforme enrichie avec cette stratégie est décrite dans la figure 8.8. Les principales différences par rapport à la version sans sauvegardes intermédiaires sont les suivantes :

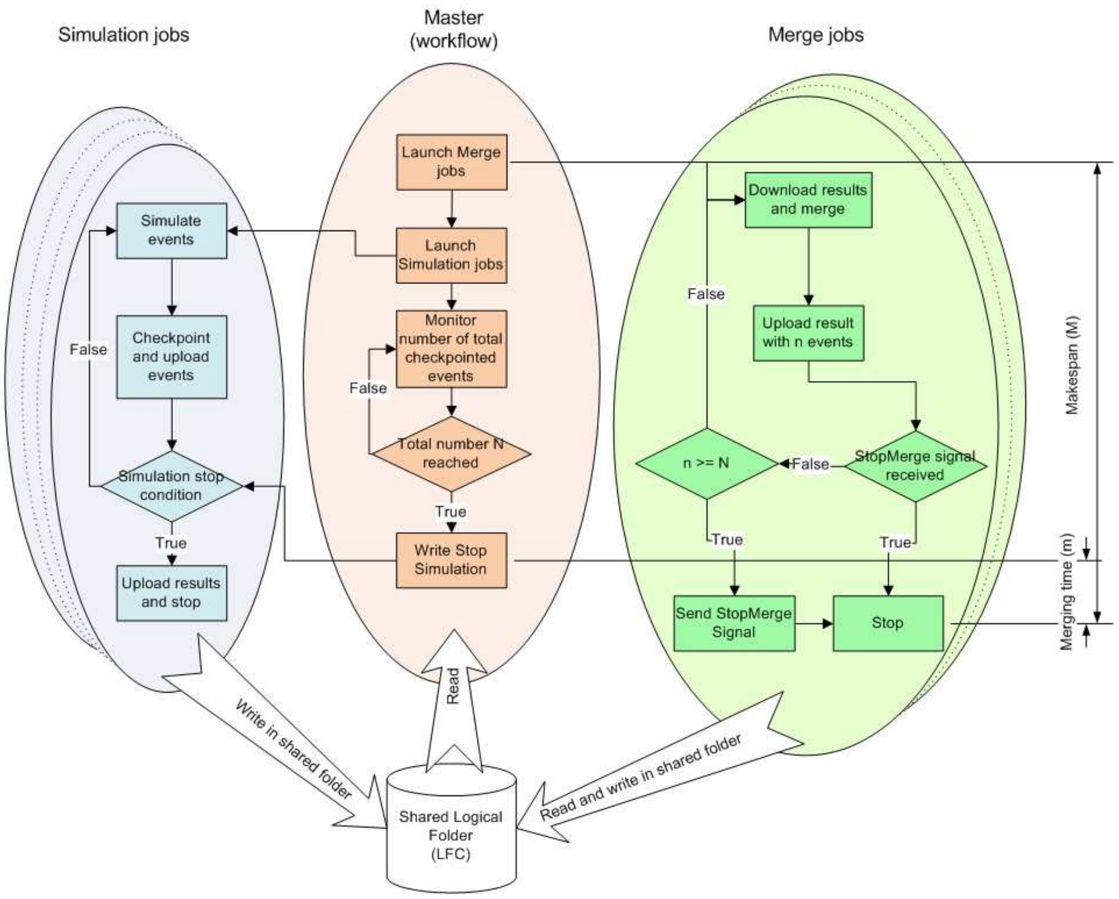


FIGURE 8.8: Algorithme avec sauvegarde intermédiaire des résultats. Les tâches de simulation sont représentées en bleu, les tâches de fusion en vert et le maître en orange. Plusieures tâches de simulation et de fusion sont exécutées en parallèle, mais, pour des raisons de lisibilité, une seule tâche de simulation et une tâche de fusion sont représentées. Les tâches de simulation chargent leur résultats régulièrement, à la fréquence des sauvegardes intermédiaires. La condition `SimulationStop` donnée par le maître entraine la fin des tâches de simulation. A partir de ce moment, uniquement les tâches de fusion continuent à s'exécuter pendant le "extra merging time". Le signal `StopMerge` est envoyé par la première tâche de fusion ayant fusionné le nombre requis d'événements.

- Les tâches de simulation sauvegardent et chargent leurs résultats partiels à la fréquence des sauvegardes intermédiaires.
- Le maître prend en compte le nombre d'événements sauvegardés à la différences des des événements simulés.
- Les tâches de fusion sont lancées dès le début, car des résultats partiels sont disponibles depuis le premier point de contrôle. En dehors de cela, les tâches de fusion restent les mêmes que dans la version sans sauvegarde intermédiaire.

Dans les deux configurations, chaque tâche de fusion contribue au résultat final en fusionnant une partie des résultats partiels. Le processus de fusion consiste à (i) sélectioner un nombre maximal nf de fichiers à fusionner, chaque fichier i contenant n_i événements, (ii) à supprimer les fichiers sélectionnés du dossier logique partagée, (iii) à télécharger et fusionner les fichiers sélectionnés, et (iv) à charger le résultat final contenant $n = \sum n_i$ événements dans le dossier logique partagé. nf est un paramètre de la plateforme et il est le même pour toutes les tâches de fusion. Pour un bon équilibrage de charge de l'activité de fusion, nf devrait être inférieur au nombre total de fichiers à fusionner divisé par le nombre total de tâche de fusion. En effet, la charge est mieux répartie en petits morceaux. Néanmoins, plus il y a des morceaux, plus le surcoût est important. Dans notre cas, si les fichiers sont fusionnés deux par deux ($nf = 2$), de nombreux résultats intermédiaires seront générés, augmentant le nombre de transferts de fichiers et, par conséquent, le temps de transfert total. Dans ce qui suit, nf sera fixé expérimentalement.

Etant donné que le processus de fusion est commutatif et associatif, un résultat fusionné est du même type que les entrées de l'opération de fusion et il peut être re-fusionné comme tout autre résultat partiel. La seule contrainte est que le même fichier ne doit pas être fusionné à plusieurs reprises. Un mécanisme de verrouillage est utilisé afin que cette condition soit respectée.

8.3.2 Modèle

La charge de travail Γ d'une simulation Monte-Carlo est librement divisible et, conformément à la Section 8.2, l'algorithme d'équilibrage de charge dynamique se comporte comme si des tâches d'un événement sont distribués en permanence à n tâches parallèles. Dans ce cas, les tâches de simulation terminent quasi simultanément, comme on peut le voir sur le flot d'exécution sur la Figure 4.1. Dans ces conditions, si on ignore les erreurs d'exécution et la variabilité de la latence, le makespan M peut être modélisé comme le temps d'attente moyen plus la durée moyenne des tes temps d'exécution des n tâches parallèles : $\mathrm{M} = \frac{\Gamma}{n} + E_L$ comme expliqué dans [Mościcki et al., 2011].

Si l'on tient compte du fait que, avec un taux d'échec ρ, seulement $n(1-\rho)$ tâches contribuent à la simulation, le modèle devient : $\Gamma = n(1-\rho)(M - E_L)$, c'est à dire $M = \frac{\Gamma}{n(1-\rho)} + E_L$

Et si l'on considère le temps de fusion m en plus du temps de calcul, nous avons :

$$M = \frac{\Gamma}{n(1-\rho)} + m + E_L \tag{8.1}$$

Dans ce qui suit, la valeur du temps de fusionner m sera mesurée à partir d'expériences réelles.

Si la sauvegarde partielle des résultats est activée, les tâches qui échouent peuvent encore contribuer au résultat final avec les résultats sauvegardé avant l'échec. Dans ce cas, le makespan dépend de F, la fonction cumulative des probabilités (CDF) d'échec (time to failure TTF) sur l'infrastructure cible. $F(t)$ est la probabilité qu'une tâche ne renvoie par d'erreur pendant une durée t.

Soit c la période des points de reprise d'un tâche de simulation. Nous supposons que c est fixe et ne peut être modifié au cours de la simulation. Comme les tâches de simulation ne commencent pas simultanément en raison de leurs temps d'attente individuels, leurs sauvegardes partielles ne sont pas synchronisés non plus. La fin de la simulation est donnée par la dernière sauvegarde de la tâche qui contribue suffisamment pour atteindre la condition d'arrêt, tandis que les autres tâches sont encore en train de calculer.

La durée totale Γ du CPU consommé par la simulation correspond à la somme des temps de calcul des résultats sauvegardés par des tâches de simulation :

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

Ainsi :

$$M \quad = \quad \frac{\Gamma}{n\left(1 - \frac{F(kc)+F(c)}{2}\right)} + \frac{c}{2} + m + E_L \tag{8.2}$$

8.3.3 Expériences et résultats

Les deux scenarii, avec et sans sauvegardes intermédiaires ont été mis en oeuvre en utilisant MOTEUR et sont intégrés dans le portail web VIP décrit dans [Glatard et al., 2012]. Les tâches sont exécutées à l'aide des tâches pilotes Dirac sur les ressources disponibles pour l'organisation virtuelle (VO) biomed dans le cadre d'EGI. Les expériences ont été menées avec une simulation de protonthérapie en utilisant GATE.

Trois expériences ont été menées :

La première expérience (Exp 1) vise à démontrer l'importance des sauvegardes intermédiaires pour des simulations très longues. Pour cette expérience, nous avons exécuté GATE avec 440 millions d'événements représentant environ un an de temps CPU. Lors d'une exécution avec 300 tâches de calcul, la durée d'une tâche est supérieure à 24 heures. Nous avons mesuré le nombre d'événements simulés et fusionnés avec et sans sauvegardes intermédiaires.

La deuxième expérience (Exp 2) vise à déterminer l'impact de plusieurs tâches de fusion parallèles. Pour cette expérience, nous avons effectué une simulation GATE avec 50 millions d'événements représentant environ 41 jours de CPU. L'implémentation sans point de contrôle a été utilisée. Quatre séries d'exécutions ont été réalisées, avec 1, 5, 10 et 15 tâches de fusion.

La troisième expérience (Exp 3) étudie l'influence des sauvegardes intermédiaires des résultats. Comme pour la seconde expérience, nous avons effectué une simulation GATE vec 50 millions d'événements représentant environ 41 jours de CPU. Pour cette expérience, l'implémentation avec sauvegardes intermédiaires a été utilisée. Trois séries d'exécution ont été réalisées, avec une période de sauvegarde partielle de 30, 60 et 120 minutes. Dans chaque cas, 10 tâches de fusion parallèles ont été lances dès le début de la simulation.

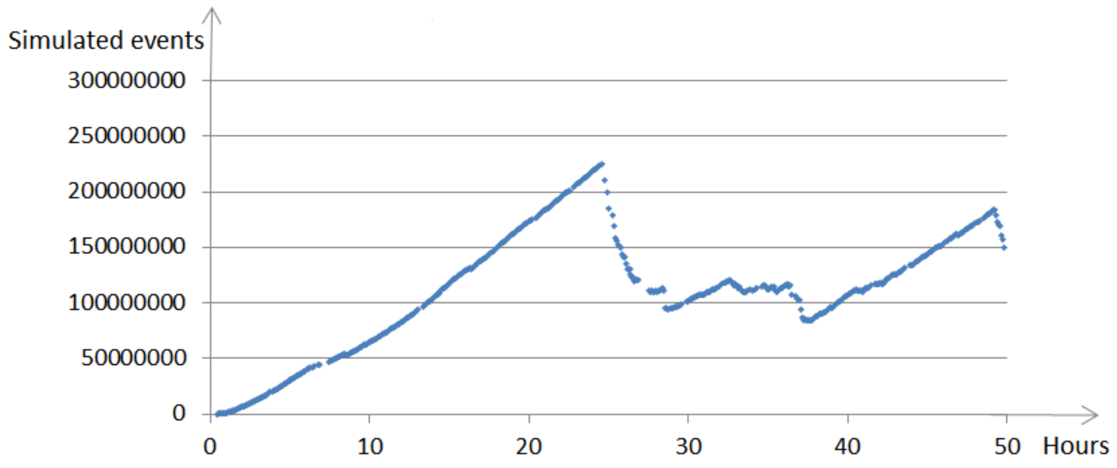
Pour les deux dernières expériences, nous avons mesuré le temps CPU total de la simulation (Γ), le temps d'attente moyen des tâches (E_L), le temps de fusion (m), le taux d'échec des tâches de simulation (ρ) et la proportion des tâches de simulation qui a échouée avant le premier point de contrôle ($F(c)$) pour les simulations avec des sauvegardes intermédiaires.

Pour les trois expériences, la simulation GATE a été divisée en 300 tâches et chaque tâche de fusion a sélectionné un maximum de 10 fichiers ($nf = 10$) à chaque étape de fusion. Les expériences ont été répétées trois fois pour capter une partie de la variabilité de la grille.

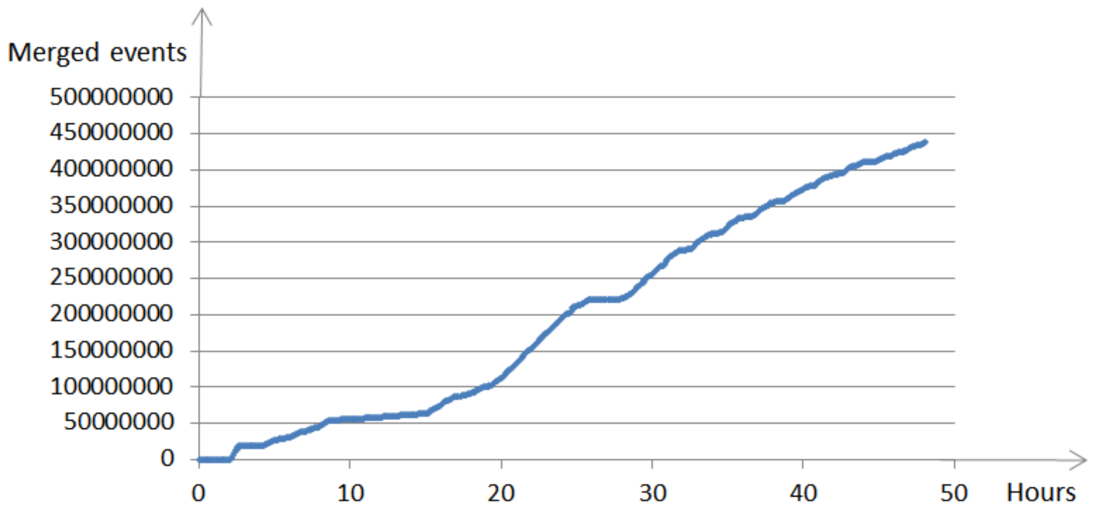
8.3.3.1 Valeur ajoutée des sauvegardes intermédiaires (Exp 1)

La Figure 8.9 indique le nombre d' événements fusionnés et simulés au cours du temps. Les événements fusionnés sont des événements qui ont été traités au moins une fois par une fusion, tandis que les événements simulés résident toujours sur le disque local d'une tâche de simulation et ne sont pas disponibles pour la fusion dans le cas où la tâche de simulation échoue. Pour l'implémentation sans sauvegarde intermédiaire (Figure 8.9-a) le nombre de résultats simulés augmente régulièrement au cours des premières 24 heures. Ensuite, une partie des tâches est tuée par des sites qui imposent un temps d'exécution maximal. Si les événements simulés ne sont pas sauvegardés et chargés périodiquement, ils sont perdus lorsque les tâches sont tuées. Les tâches tuées sont resoumises, ce qui explique pourquoi le nombre d'événements augmente encore même après les premières 24 heures . Dans cette expérience, l'implémentation sans sauvegarde intermédiaire n'est tout simplement pas en mesure de compléter la simulation de 440 millions événements.

Inversement, l'implémentation avec sauvegardes intermédiaires (figure 8.9-b) est en mesure de compléter la simulation. Vu que les événements sauvegardés et chargés périodiquement ne sont pas perdus lorsque les tâches sont tuées, leur nombre augmente régulièrement jusqu'à la fin de la simulation.



(a) Implémentation sans sauvegarde intermédiaire



(b) Implémentation avec sauvegardes intermédiaires

FIGURE 8.9: Résultats de l'expérience avec une simulation très longue (Exp 1), représentant environ un an de temps CPU. L'implémentation sans sauvegarde intermédiaire (a) n'est pas en mesure de terminer la simulation parce que les événements des tâches tuées sont entièrement perdus. Pour l'implémentation avec sauvegardes intermédiaires (b), le nombre d'événements augmente de façon constante jusqu'à la fin de la simulation.

Exécution	ρ	E_L (s)	m (s)	M réel (s)	M modèle (s)	Erreur du modèle (%)
1 tâche de fusion #1	0.148	5298	19980	41916	38653	7.8
1 tâche de fusion #2	0.317	1404	26760	42460	46166	-8.7
1 tâche de fusion #3	0.180	1554	5220	21528	21490	0.2
Moyenne	-	-	17320	35301	35436	-
5 tâches de fusion #1	0.187	1361	3060	18384	20159	-9.7
5 tâches de fusion #2	0.191	2295	6480	23742	24718	-4.1
5 tâches de fusion #3	0.140	950	9120	25613	23092	9.8
Moyenne	-	-	6220	22579	22656	-
10 tâches de fusion #1	0.102	1346	1920	17601	16588	5.8
10 tâches de fusion #2	0.171	2143	2580	21927	19749	9.9
10 tâches de fusion #3	0.213	3240	2940	23055	22176	3.8
Moyenne	-	-	2480	20861	19504	-
15 tâches de fusion #1	0.128	2369	4080	21637	21061	2.7
15 tâches de fusion #2	0.123	2483	3060	20343	19659	3.4
15 tâches de fusion #3	0.150	1580	2460	18326	18406	-0.4
Moyenne	-	-	3200	20102	19709	-

TABLE 8.1: Résultats des expériences avec des tâches de fusion multiples (Exp 2).

8.3.3.2 Impact des tâches de fusion multiples (Exp 2)

Le tableau 8.1 détaille les mesures obtenues lors de la deuxième expérience (Exp 2). L'erreur du modèle est calculée comme suit : $(Mreal - Mmodel)/Mreal * 100$. Le modèle explique correctement les expériences, avec une erreur relative inférieure à 10%, et moins de 5% pour la moitié des simulations.

Les résultats montrent que l'utilisation d'une tâche unique de fusion est clairement sous-optimale. D'après le tableau 8.1, le makespan moyen avec une tâche de fusion peut être réduit de 40% en utilisant 10 tâches de fusion parallèles (de 35301 à 20861 secondes). Cela est dû à une diminution importante du temps de fusion, ce qui peut représenter plus de 50% du makespan total des simulations avec une seule tâche de fusion, alors qu'elle représente moins de 15% pour les simulations avec 10 tâches de fusion parallèles.

Nous remarquons aussi qu'il y a un seuil au-dessus duquel l'augmentation du nombre de tâches de fusion n'est pas utile. Dans notre cas, les expériences avec 10 tâches de fusion montrent des performances similaires aux expériences avec 15 tâches de fusion.

8.3.3.3 Influence de la période des sauvegardes intermédiaires (Exp 3)

Le tableau 4.3 montre les mesures obtenues lors de la troisième expérience (Exp 3), ainsi que les valeurs du modèle calculées en utilisant l'équation 8.2, en supposant que

Exécution	ρ	$F(c)$	E_L (s)	m (s)	M réel (s)	M modèle (s)	Erreur du modèle (%)
30 min #1	0.261	0.239	1988	3060	21803	23968	-9.7
30 min #2	0.239	0.211	1026	7680	25214	27372	-9.1
30 min #3	0.202	0.184	3643	2640	23470	23794	-0.7
Moyenne	-	-	-	4460	23495	25024	-
60 min #1	0.183	0.177	1343	1080	20056	20744	-3.5
60 min #2	0.196	0.164	2501	840	24163	21724	10.6
60 min #3	0.206	0.193	1453	1860	22060	23584	-6.2
Moyenne	-	-	-	1260	22093	21972	-
120 min #1	0.120	0.100	4319	720	26071	27872	-6.4
120 min #2	0.263	0.241	4879	900	26368	25813	1.6
120 min #3	0.279	0.250	5336	900	28378	27241	3.9
Moyenne	-	-	-	840	26939	26990	-

TABLE 8.2: Results des expériences avec 10 tâches de fusion et sauvegarde intermédiaire des résultats (Exp 3).

$F(kc) = \rho$. Globalement, le modèle explique correctement les expériences, avec une erreur relative inférieure à 10%.

D'après le tableau 8.2, le temps de fusion m diminue à mesure que la période de sauvegarde augmente. Cela peut s'expliquer par le fait que les tâches de fusion peuvent être saturées avec des résultats partiels si les sauvegardes intermédiaires sont trop fréquentes. En même temps, selon le modèle et l'équation 8.2, le makespan augmente avec la durée de la période de sauvegarde. Un compromis est donc nécessaire. Parmi les trois points, on remarque que la période de 60 minutes offre le meilleur makespan moyen.

Les expériences en production garantissent que toutes les hypothèses sont réalistes, mais elles limitent également la reproductibilité. Pour remédier à cette limitation, la section suivante visera à reproduire ces expériences de production dans un environnement de simulation basé sur l'environnement de simulation SimGrid. Afin d'assurer des simulations réalistes, (i) nous implémentons notre plateforme dans l'environnement de simulation, (ii) nous paramétrons la simulation à l'aide des traces capturées à partir de nos expériences réelles, et (iii) nous validons la performance des résultats obtenus en simulation en s'appuyant sur les résultats réels.

8.4 Simulation de l'exécution d'une application et des services déployés sur EGI

Le but de cette section est d'étudier comment les exécutions des applications déployées sur EGI peuvent être reproduites et analysées à travers la simulation. Le but ultime de cette étude est de pouvoir rejouer les exécutions réelles lancées à partir de la plateforme

VIP afin de comprendre, étudier et améliorer leur performance. Nous ne cherchons pas à simuler l'ensemble du système EGI, mais seulement le sous-ensemble de la plateforme et des services utilisés par une exécution particulière d'une application.

A partir de notre expérience du système réel, nous construisons un simulateur basé sur l'environnement de simulation SimGrid [Casanova et al., 2008] pour simuler la plateforme matérielle, les services de base, le déploiement et l'application. La plateforme est l'ensemble des ressources matérielles utilisées par l'application : le stockage, le réseau et les CPU. Les services sont des processus logiciels qui sont réutilisés par différentes applications, par exemple, un moteur de workflow, un ordonnanceur (job scheduler) ou un catalogue de fichiers. Le déploiement correspond à la mise en correspondance entre les services logiciels et les entités de la plateforme. Enfin, l'application englobe tous les processus spécifiques à un calcul effectué par un utilisateur.

La plateforme est supposée déjà modélisée, et nous nous concentrons sur les services, le déploiement et l'application. Les différences entre les platesformes réelle et simulée, en particulier les CPU et les caractéristiques du réseau, sont compensées par la calibration de l'application simulée sur la plateforme de simulation.

Il existe une variété de boîtes à outils de simulation, dont OptorSim [Bell et al., 2003], GridSim [Buyya and Murshed, 2002], PeerSim [Montresor and Jelasity, 2009], CloudSim [Calheiros et al., 2011], et SimGrid [Casanova et al., 2008, Bobelin et al., 2012]. Nous avons choisi d'utiliser SimGrid parce que c'est un projet actif qui expose, via des API bien documentées, un ensemble d'outils de simulation riche, facile et complet. SimGrid fournit les fonctionnalités de base pour la simulation d'applications dans des environnements distribués et hétérogènes. Dans ce qui suit, nous décrivons les fonctionnalités utilisées dans notre simulation.

8.4.1 Conception de la simulation

8.4.1.1 Plateforme

Malgré plusieures tentatives de modélisation de la plateforme EGI, il n'existe actuellement aucun modèle approprié disponible pour nos simulations. Ceci est principalement dû au manque d'informations détaillées sur (i) la configuration (nombre et les performances des unités de calcul, de bande passante intra-site, etc.) de ses plus de 300 sites distribués et (ii) les connexions du réseau inter-site. Il est particulièrement difficile de recueillir l'information nécessaire est car elle change souvent.

Par conséquent, nous avons utilisé le modèle de la plateforme Grid'5000 [3]. Il comprend 10

3. https://www.grid5000.fr

sites (AS), 40 clusters, approximativement 1500 noeuds, et l'infrastructure réseau de la plateforme Grid'5000. Afin d'y rajouter des éléments de stockage d'une manière similaire à EGI, nous avons ajouté un hôte par site de la plate-forme.

8.4.1.2 Services

Deux processus SimGrid ont été mis en oeuvre pour simuler (i) le catalogue de fichiers, LFC, et (ii) des éléments de stockage, SEs. Ils simulent les communications impliquées dans les opérations de transfert de fichiers en utilisant l'API MSG de SimGrid. Les principaux messages traités par notre LFC simulé sont l'enregistrement de fichiers, la gestion des dossiers, et le listing des réplicas. Une seule réplique par fichier peut être manipulée à l'heure actuelle. Notre SE dispose de deux opérations : le téléchargement et le chargement des fichiers.

Les transferts de fichiers sont implémentés comme des tâches avec une quantité de calcul nulle envoyées par la source de transfert vers la destination. Les opérations de lecture et écriture sur les disques de stockage sont supposées négligeables par rapport aux transferts inter-sites.

Un processus maître et un processus esclave ont été mis en oeuvre pour simuler la création des tâches, l'ordonnancement et l'exécution de la charge de travail, comme illustré sur la figure 8.10. Le maître simule à la fois le moteur de workflow MOTEUR et l'ordonnanceur Dirac. Il initialise d'abord les esclaves en leur envoyant une tâche `init` avec une quantité de calcul et de transfert de données nulles. Il génère ensuite les tâches Monte-Carlo et les envoie aux esclaves dès qu'ils répondent à la tâche `init` (premier arrivé, premier servi) . Si la sauvegarde intermédiaire des résultats est activée, le maître envoie les tâches de fusion dès le début de la simulation ; sinon, il attend d'abord que les tâches Monte-Carlo soient finies.

Les esclaves simulent à la fois les tâches pilotes Dirac et les tâches de l'application. Ils se déclarent au maître en envoyant un message `ack` en réponse à `init` . Ils peuvent traiter aussi bien des tâches Monte-Carlo que des tâches de fusion.

8.4.1.3 Déploiement

Le processus d'approvisionnement des ressources, réalisé dans le système réel par l'ordonnanceur WMS de gLite, les files d'attente et l'ordonnanceur de tâches pilotes DIRAC, est simulé ici en générant de façon aléatoire la liste d'hôtes utilisés pour déployer les esclaves pour chaque simulation. Deux listes d'hôtes différentes sont utilisées, l'une pour les tâches Monte-Carlo et l'autre pour les tâches de fusion. Les processus maître et

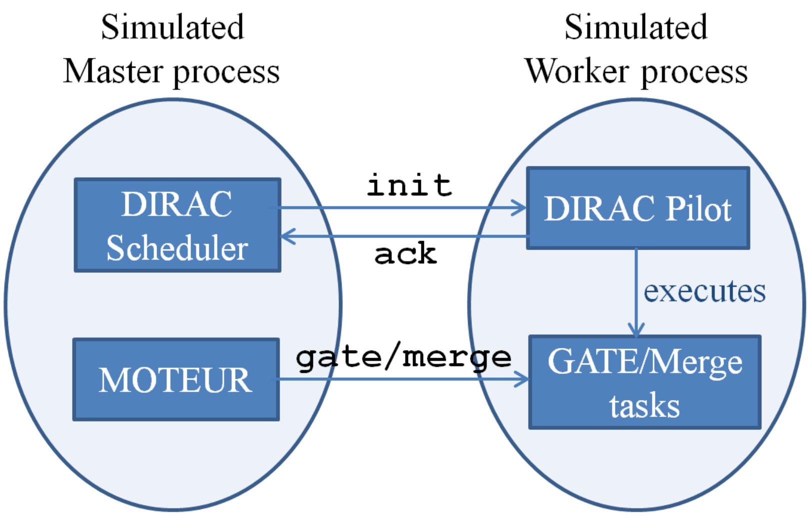


FIGURE 8.10: Processus maître et esclave simulés avec SimGrid.

Hôte simulé	Fichier de description	Valeurs possibles	Source des valeurs
Puissance	Fichier plateforme	4.E9 - 30E9 flops/s	Description de Grid5000
Latence	Fichier de disponibilité	0 jusqu'à t, 1 ensuite	Latence t extraite des traces réelles
Erreur	Fichier d'état	ON jusqu'à t, OFF ensuite	Temps d'erreur t extrait des traces réelles

TABLE 8.3: Propriétés de l'hôte en simulation et les détails de mise en oeuvre. Les hôtes en simulation récupèrent leur puissance de calcul à partir du fichier de description de la plateforme Grid'5000. Les latences des tâches et les erreurs sont extraites des traces réelles et sont simulées à l'aide des fichiers de disponibilité et d'état de SimGrid.

LFC sont déployés une fois pour toutes sur des noeuds choisis aléatoirement dans la plateforme.

La latence et les erreurs des tâches sont simulées à l'aide de deux mechanismes disponibles dans SimGrid : l'état et la disponibilité de chaque hôte. Un temps de latence t est modélisé par une disponibilité de l'hôte de 0 jusqu'à l'instant t . Une erreur intervenue au moment t est modélisée par un passage de l'hôte de l'état ON à OFF au moment t. Les valeurs de latence et les temps de panne sont extraites à partir de traces réelles. Le tableau 8.3 résume les propriétés de l'hôte et les détails de implémentation. Pour faire correspondre les temps de latence/échec extraits de traces réelles aux valeurs de disponibilités/état des hôtes simulés, nous avons fait un tri par ordre decroissant des valeurs de puissance CPU comme illustré dans le tableau 8.4.

No d'hôte	CPU Grid5000(flops/s)		CPU EGI (BogoMips)		Latence (s)	Temps de panne (s)
1	23E9		4522		1440	240
2	27E9	$\downarrow$	5332	$\downarrow$	840	18000
3	30E9		5600		900	120

TABLE 8.4: Les hôtes réels et simulés sont triés en fonction de leur puissance CPU. Des exemples de latence et les temps de panne des tâches réelles sont extraits des journaux d'exécution réels et affectés aux tâches correspondantes en simulation.

8.4.1.4 Application

Deux types de tâches sont simulés : des tâches Monte-Carlo et des tâches de fusion. Les tâches Monte-Carlo téléchargent leur données d'entrée en utilisant le procédé de transfert de fichiers et démarrent ensuite la simulation Monte-Carlo. Elles signalent régulièrement leur nombre d'événements calculés au maître en envoyant des messages `events`. Si la sauvegarde intermédiaire des résultats est activée, elles transfèrent également leurs résultats au SE du site en utilisant les méthodes de transfert de fichiers. Les tâches Monte-Carlo transfèrent leur résultat final et arrêtent leur exécution quand elles reçoivent le message `stop` de la part du maître. Une taille de fichier fixe est utilisée pour tous les résultats Monte-Carlo.

Le maître reçoit des messages `events` précisant le nombre d'événements Monte-Carlo calculés par l'émetteur. Sur cette base, il maintient le nombre total d'événements calculés, et envoie le message `stop` à tous les esclaves exécutant des tâches Monte-Carlo dès que le nombre total d'événements est atteint.

8.4.1.5 Calibration

La calibration de certains paramètres est nécessaire pour compenser les écarts de performances entre les plateformes réelle et simulée. Nous utilisons la règle suivante pour déterminer le coût de calcul d'un événement Monte-Carlo :

$$x[flop/event] = \frac{\bar{p}_{sim}[flop/s]CPU_{real}[s]}{E_{real}[event]}, \tag{8.3}$$

où les unités sont données entre crochets, x est le coût de calcul d'un événement, $\bar{p}_{sim}$ est la performance moyenne des hôtes sur la plateforme de simulation, CPU_{real} est le temps de calcul cumulatif des tâches Monte-Carlo sur la plateforme réelle (EGI), et E_{real} est le nombre d'événements Monte-Carlo. Nous considérons l'exécution réelle d'un calcul GATE de 50 millions d'événements qui représente environ 61000 minutes de CPU sur les ressources de la VO biomed. Le modèle de plateforme Grid'5000 utilisé pour la simulation a une performance moyenne $\bar{p}$ de $12,68$ Gflop/s, ce qui donne $x = 0,93$ Gflop/événement.

Nous utilisons une règle similaire pour déterminer le coût de calcul d'une opération de fusion entre deux fichiers :

$$y[flop/merge] \quad = \quad \bar{p}_{sim}[flop/s]CPUm_{real}[s], \tag{8.4}$$

où $CPUm_{real}$ représente le temps CPU d'une opération de fusion sur la plateforme réelle. On obtient $y = 126$ Gflop dans le cas de notre application sur la plateforme simulée.

La taille des fichiers produits par les tâches Monte-Carlo a aussi été calibrée de façon à ce que les temps de transfert de fichiers simulés correspondent au mieux à la durée mesurée en réalité. Trois tailles de fichiers de 10, 15 et 20 Mo ont été testés et nous avons gardé la valeur de 15 Mo.

8.4.2 Validation de la simulation

Plusieures expériences one été ralisées pour comparer l'exécution réelle et simulée pour différents nombres de tâches de fusion (1, 5, 10 et 15) et différentes périodes de sauvegarde intermédiaire (30, 60, 120 minutes). Pour chaque expérience, la phase de Monte-Carlo a été exécutée par 300 tâches parallèles. Trois traces réelles sont disponibles pour chaque configuration et 5 simulations ont été effectuées pour chaque trace réelle.

La Figure 8.11 montre les makespan réels et simulés de la phase de fusion en fonction du nombre de tâches de fusion pour l'implémentation sans sauvegarde intermédiaire. Les makespan réels et simulés suivent la même tendance : le temps de fusion diminue lorsque le nombre de tâches de fusion augmente jusqu'au seuil de 10 tâches de fusion, après quoi il augmente légèrement.

Le makespan simulé est moins variable que le makespan réel pour une et cinq tâches de fusion. C'est probablement parce que la plateforme Grid'5000 utilisée dans la simulation est nettement moins hétérogène que l'EGI.

Avec une seule tâche de fusion, le temps de fusion dépend étroitement de la performance du noeud sélectionné. C'est ce qui explique la grande variabilité du makespan pour cette configuration. Lorsque le nombre de tâches de fusion augmente (10 ou 15), la performance globale est moins sensible à l'hétérogénéité puisque quelques hôtes performants pourront compléter le travail non accompli par les autres.

La Figure 8.12 montre les makespan réels et simulés de la phase de fusion en fonction de la période de sauvegarde intermédiaire pour l'implémentation avec sauvegarde intermédiaire. Les résultats de simulation sont proches des valeurs réelles et suivent la même

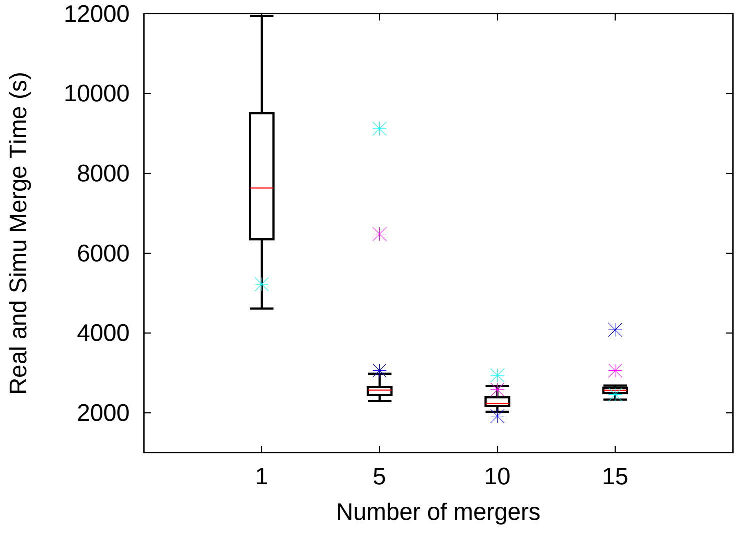


FIGURE 8.11: Makespan réels et simulés de la phase de fusion en fonction du nombre de tâches de fusion pour l'implémentation sans sauvegarde intermédiaire. Les 12 valeurs réelles sont présentées individuellement, tandis que les 60 valeurs simulées sont regroupées en quatre boîtes à moustaches avec des valeurs médianes, une pour chaque nombre de tâches de fusion. En raison des problèmes d'échelle, seulement 1 valeur réelle est affichée pour la configuration avec une tâche de fusion, les deux autres sont à 19980 et 26760 s.

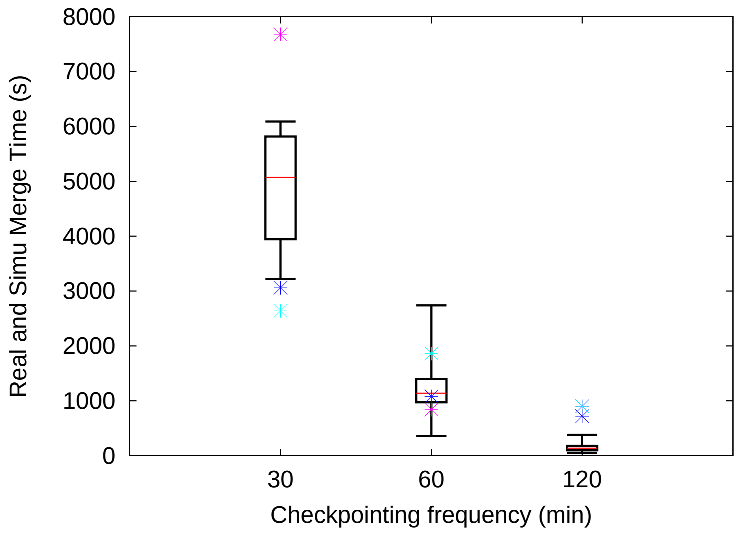


FIGURE 8.12: Makespan réels et simulés de la phase de fusion comme en fonction de la période de sauvegarde intermédiaire pour l'implémentation avec sauvegarde intermédiaire. Les valeurs réelles sont présentées individuellement, tandis que les 45 valeurs de simulation sont regroupées en trois boîtes à moustaches avec valeurs médianes, une pour chaque période de sauvegarde intermédiaire.

tendance : le temps de fusion diminue lorsque la période augmente car le nombre de résultats partiels de fusion diminue. Pour la configuration avec une période de sauvegarde intermédiaire de 30 minutes, la performance de l'application réelle est très variable. Il s'agit d'un cas limite pour lequel l'hétérogénéité de la plateforme joue un rôle important. La simulation parvient à capturer ces différences seulement dans une certaine mesure, les valeurs de simulation montrant une plus grande variabilité que pour les périodes de sauvegarde plus élevés.

8.5 Conclusion et perspectives

Dans le contexte complexe des systèmes distribués et hétérogènes, notre objectif principal était de proposer de nouvelles stratégies pour une exécution plus rapide et plus fiable des applications Monte-Carlo. Pour atteindre cet objectif, nous avons proposé un nouvel algorithme d'équilibrage de charge dynamique pour la phase de calcul, ainsi que des stratégies de fusion avancées utilisant de tâches de fusions multiples et la sauvegarde intermédiaire des résultats. Nous avons validé nos contributions à l'aide de multiples approches, y compris l'expérimentation, la modélisation et la simulation. En outre, nous avons intégré certaines de nos contributions dans la plateforme de production VIP, à travers laquelle elles sont utilisées par des centaines d'utilisateurs du portail.

La Section 8.2 a présenté une approche de parallélisation basée sur l'emploi des tâches pilotes et sur un nouvel algorithme d'équilibrage de charge dynamique. L'algorithme consiste en une boucle "do-while" sans partitionnement initial. Le maître somme régulièrement le nombre d'événements simulés et envoie des signaux d'arrêt aux pilotes lorsque le nombre total d'événements demandés est atteint. Les ressources de calcul sont exploitées jusqu'à la fin de la simulation et sont libérées presque simultanément. Ainsi, notre algorithme d'équilibrage de charge peut être considéré comme optimal en termes d'utilisation des ressources et de makespan. Par conséquent, nous considérons que l'approche dynamique proposée résout le problème d'équilibrage de charge pour les applications Monte-Carlo exécutées en parallèle sur des systèmes distribués et hétérogènes. Néanmoins, cette approche présente certaines limites puisque l'algorithme proposé : (i) peut conduire à calculer un peu moins ou plus d'événements que prévu initialement demandée par l'utilisateur, (ii) rend impossible la reproduction exacte d'une simulation Monte-Carlo et (iii) ne s'applique qu'aux applications Monte-Carlo pour lesquelles tous les événements sont strictement équivalents et peuvent être simulés dans n'importe quel ordre. Au-delà de la résolution de ces limitations, le travail futur pourrait consister à étudier la possibilité d'utiliser cet algorithme d'équilibrage de charge dynamique pour des applications non Monte-Carlo.

La section 8.3 a complété les optimisations proposées auparavant avec des stratégies de fusion avancées à l'aide de tâches de fusion multiples et de la sauvegarde intermédiares des résultats. Pour évaluer la nécessité des stratégies proposées, ainsi que leurs performances, trois expériences ont été menées sur EGI avec GATE. La première expérience met en évidence la nécessité d'utiliser des sauvegardes intermédiares pour les simulations très longues. La seconde fait varier le nombre des tâches de fusion parallèles et montre que l'utilisation d'une tâche de fusion unique est clairement sous-optimale. La troisième expérience montre que les sauvegardes intermédiaires des résultats peuvent être utilisées pour réaliser une fusion incrémentale et pour améliorer la fiabilité sans pénaliser le makespan. Un modèle a été proposé pour expliquer les mesures faites en production. Le modèle n'est pas prédictif, mais il donne une bonne interprétation des paramètres influençant le makespan. Les résultats expérimentaux sont conformes au modèle avec une erreur relative inférieure à 10%. Même si elles ont prouvé leur efficacité, la fusion multiple et sauvegarde intermédiaire ne sont pas encore en production. Enrichi en fonctionnalités, le workflow complet est plus complexe, plus instable et plus difficile à déboguer. Au-delà de ces questions de "mise en oeuvre", un problème plus fondamental doit encore être résolu : le bon réglage du nombre des tâches de fusion et de la période de sauvegarde.

La section 8.4 a proposé une simulation de bout en bout de la plateforme complète. En utilisant l'environnement de simulation SimGrid, nous avons simulé le déploiement des tâches pilotes, les principaux services gLite (catalogue de fichiers et l'élément de stockage) et l'exécution de notre application. Pour remédier aux disparités de performance entre les plates-formes réelle et simulée, un certain nombre de paramètres ont été calibrés à partir de traces réelles. Le simulateur résultant a été utilisé pour étudier l'influence du nombre de tâches de fusion dans le processus d'application et la période de sauvegarde intermédiaire des résultats. Les résultats montrent que les tendances observées en production sont correctement reproduites par la simulation.

Ces résultats ouvrent la voie à une évaluation meilleure et plus rapide. Néanmoins, il est discutable de penser que la simulation seule peut remplacer les autres méthodes de validation, comme l'expérimentation en production. En s'appuyant sur notre expérience, nous croyons que les deux méthodes devraient coexister car elles se complètent mutuellement. La calibration en utilisant les résultats des expériences en production peut jouer un rôle essentiel dans le réalisme de la simulation. De plus, sans une validation approfondie par rapport aux résultats réels, la simulation peut être trompeuse. Les traces des expériences en production peuvent aider à calibrer et valider la simulation, qui, une fois validée, peut être utilisée pour produire de nouveaux résultats. Dans notre cas, la calibration était indispensable car nous n'avions pas de modèle de plateforme approprié disponible pour EGI. Afin d'améliorer la description de la plateforme de simulation, deux approches pourraient être étudiées : (i) la calibration des platesformes existantes, telles que présentées ici ou (ii) la construction d'un modèle de plateforme EGI à partir des traces de production. Les travaux futurs pourraient ainsi étudier cette deuxième approche.

Compte tenu du développement croissant et de la popularité des technologies de cloud computing, il serait intéressant de mettre en oeuvre les algorithmes proposés ici sur des architectures de cloud computing. Cela impliquerait que les tâches Monte-Carlo, ainsi que les tâches de fusion s'exécutent sur les ressources de cloud computing et, par conséquent, que les résultats soient stockés sur les ressources de stockage du cloud. L'objectif principal des stratégies présentées ici était d'améliorer le makespan et la fiabilité globale. Lors d'un déplacement vers le cloud, les objectifs peuvent évoluer et les optimisations peuvent devenir plus complexes puisque le coût doit également être pris en compte. Dans ce contexte, on peut se poser la question de l'optimalité de l'algorithme d'équilibrage de charge dynamique proposé par rapport aux coûts ou d'autres paramètres pertinents pour l'utilisation du cloud. Les stratégies de fusion devront également être adaptées aux particularités du cloud, où les transferts de données peuvent être particulièrement coûteux. Néanmoins, elles pourraient également bénéficier des ressources de cloud computing plus localisées, ainsi que des systèmes de fichiers fournis dans le cloud, comme Hadoop.

Annexe A

Further results for chapter 5

Figures A.1, A.2, A.3, A.4, A.5, A.6 present further results for Chapter 5, Section 5.6. They show computing, merging and total makespans obtained in simulation as a function of the checkpointing period with the following values : 20min, 25min, 30min, 35min, 40min, 45min, 60min, 75min 90min, 105min and 120min. Each graph uses one trace corresponding to one real execution with checkpointing. In total, 6 traces are presented ; the 3 others are available in Chapter 5, Section 5.6. For each trace we executed 55 simulations, i.e. 5 repetitions (each with a different simulation deployment file) of each of the 11 checkpointing periods.

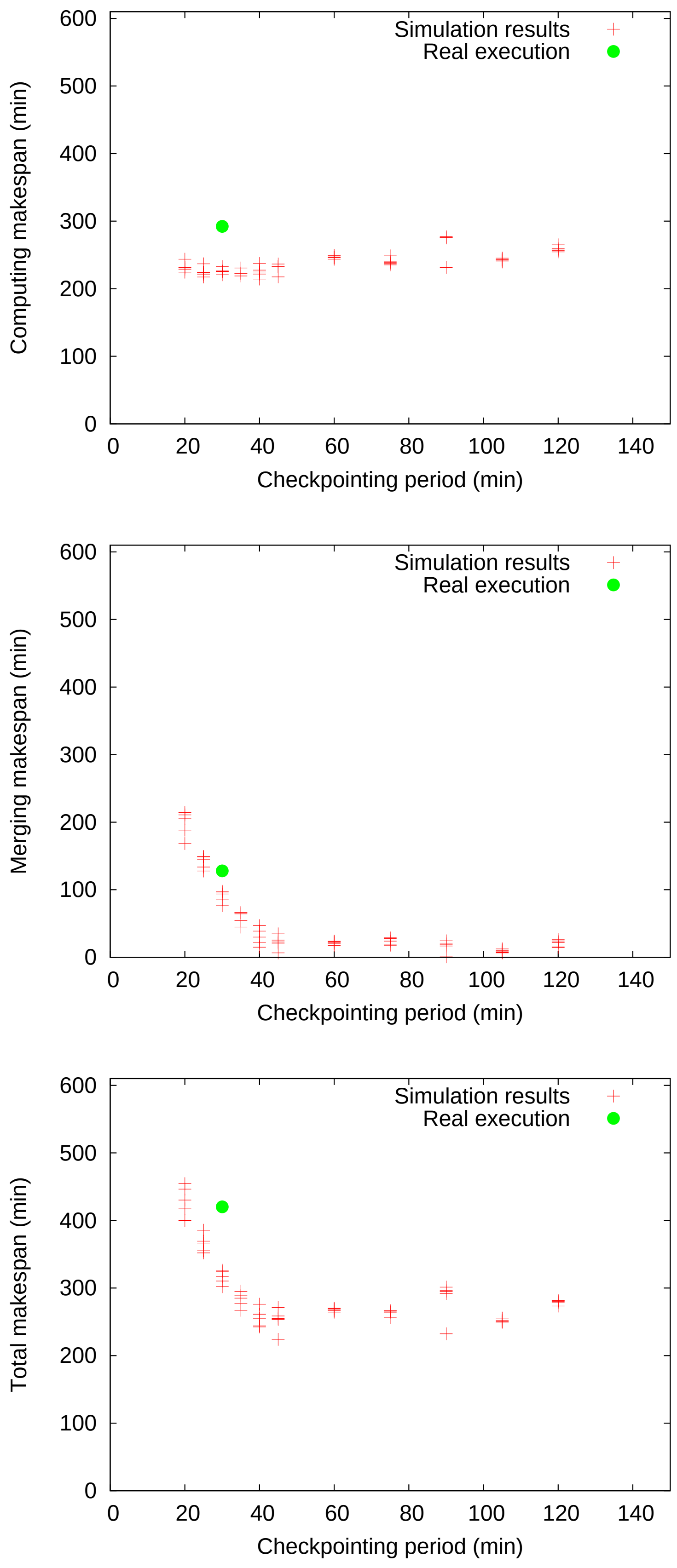


FIGURE A.1: Computing, merging and total makespans as a function of the checkpointing period.

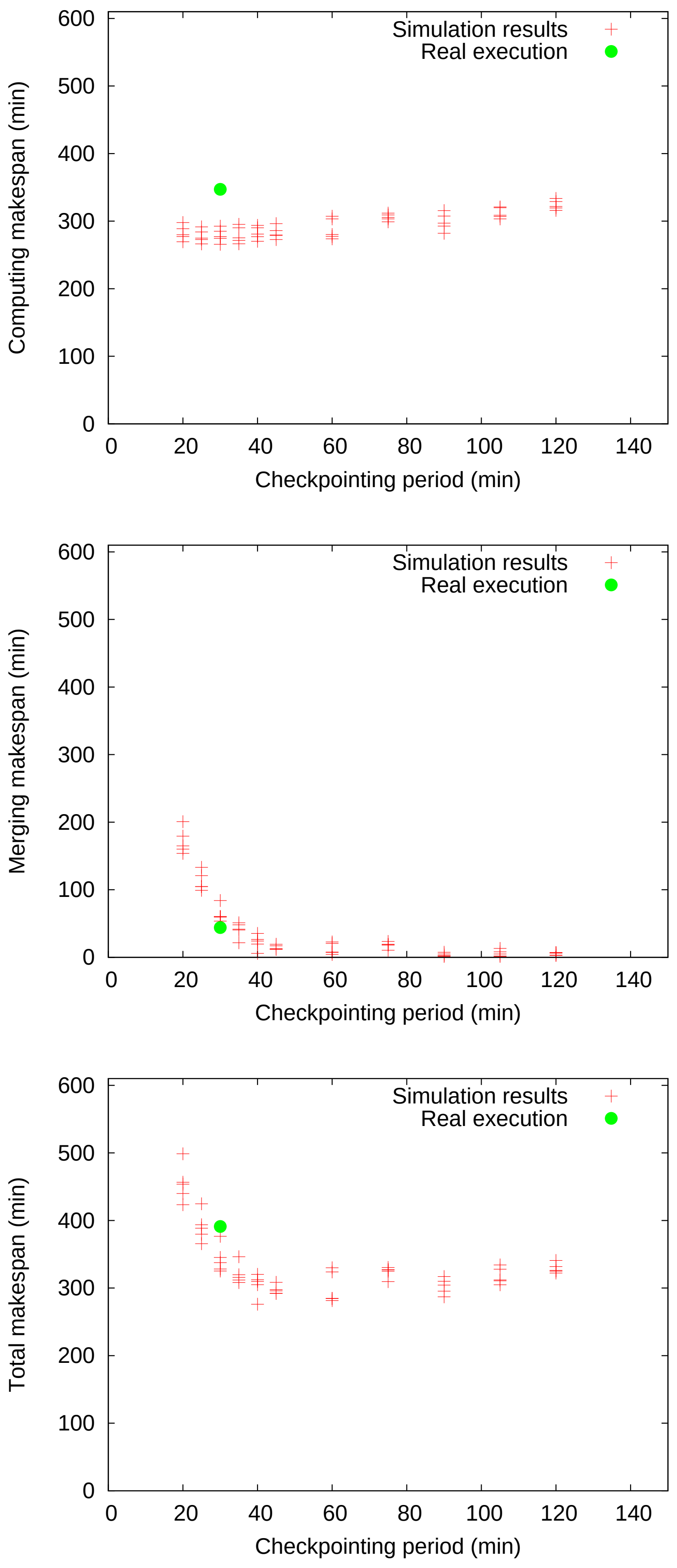


FIGURE A.2: Computing, merging and total makespans as a function of the checkpointing period.

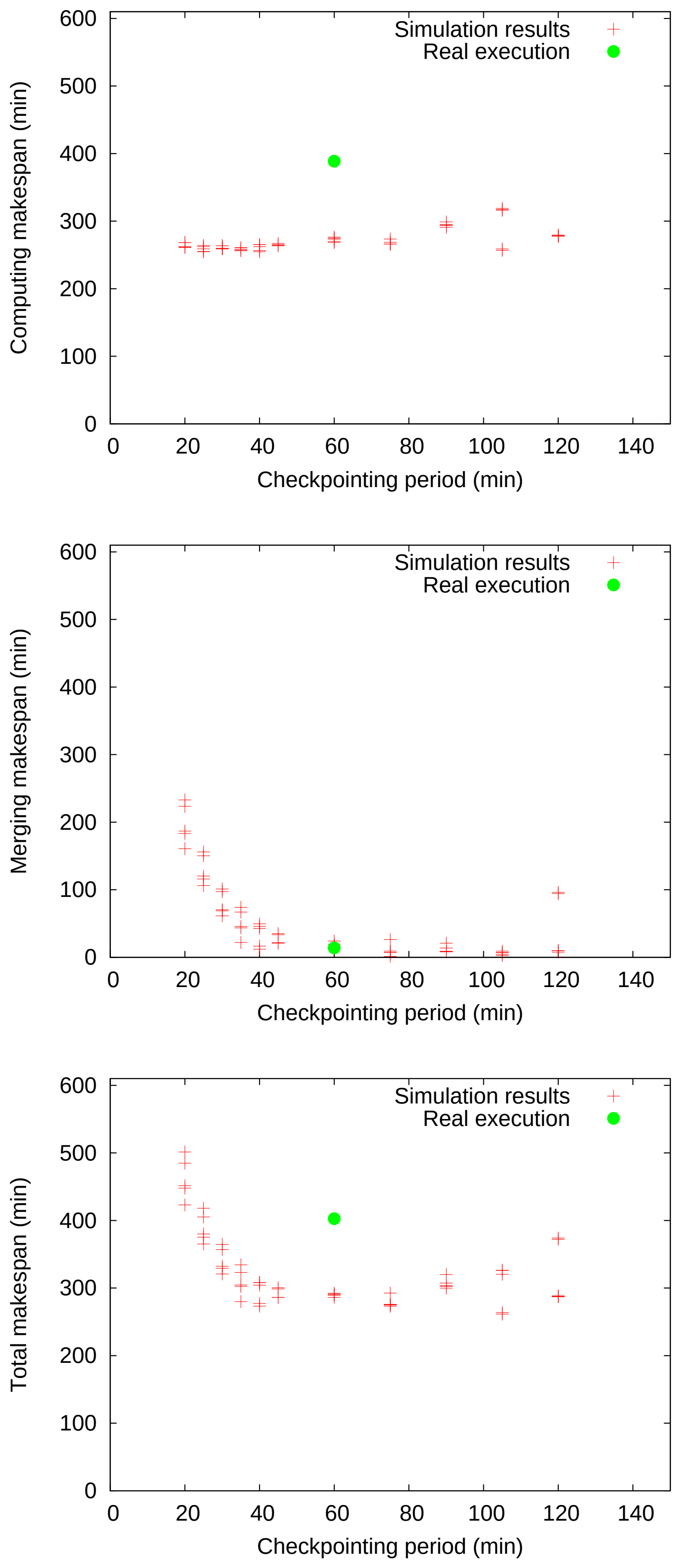


FIGURE A.3: Computing, merging and total makespans as a function of the checkpointing period.

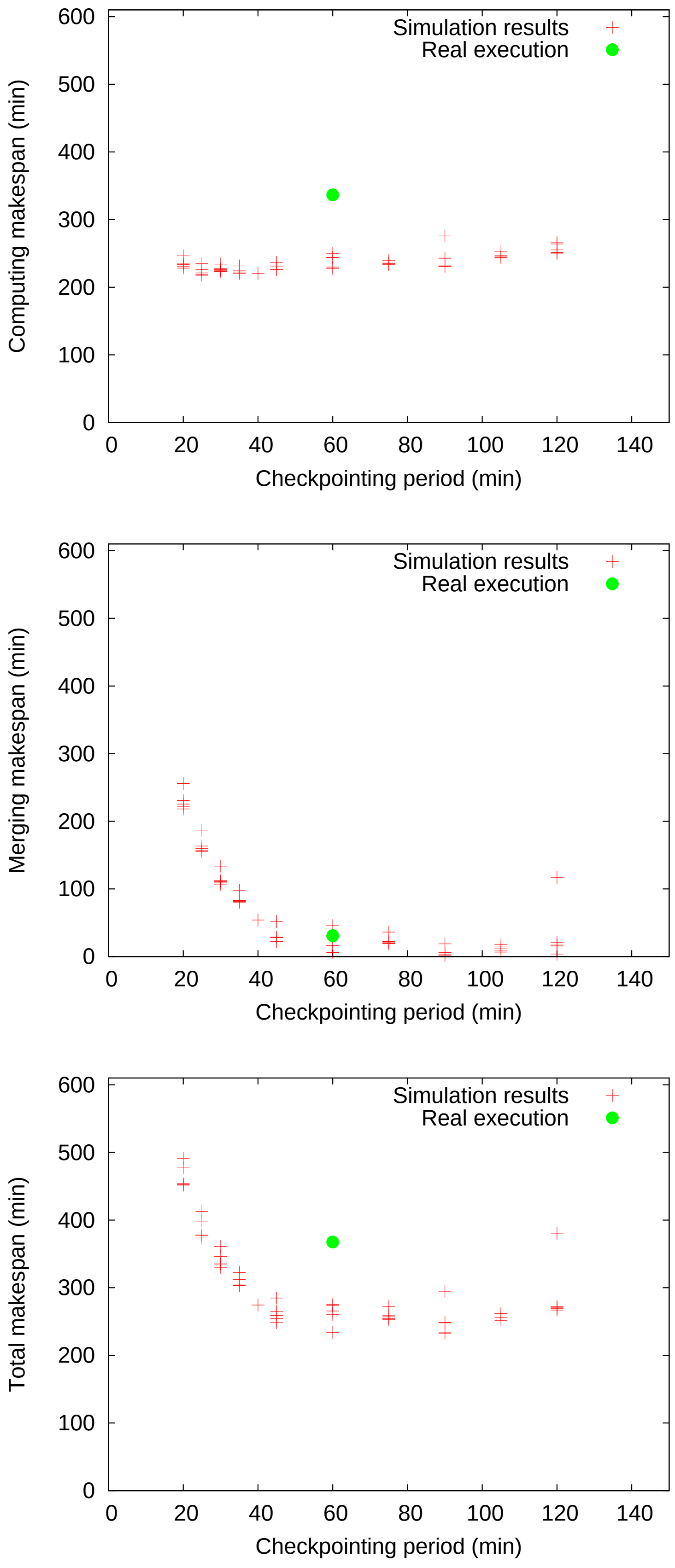


Figure A.4: Computing, merging and total makespans as a function of the checkpointing period.

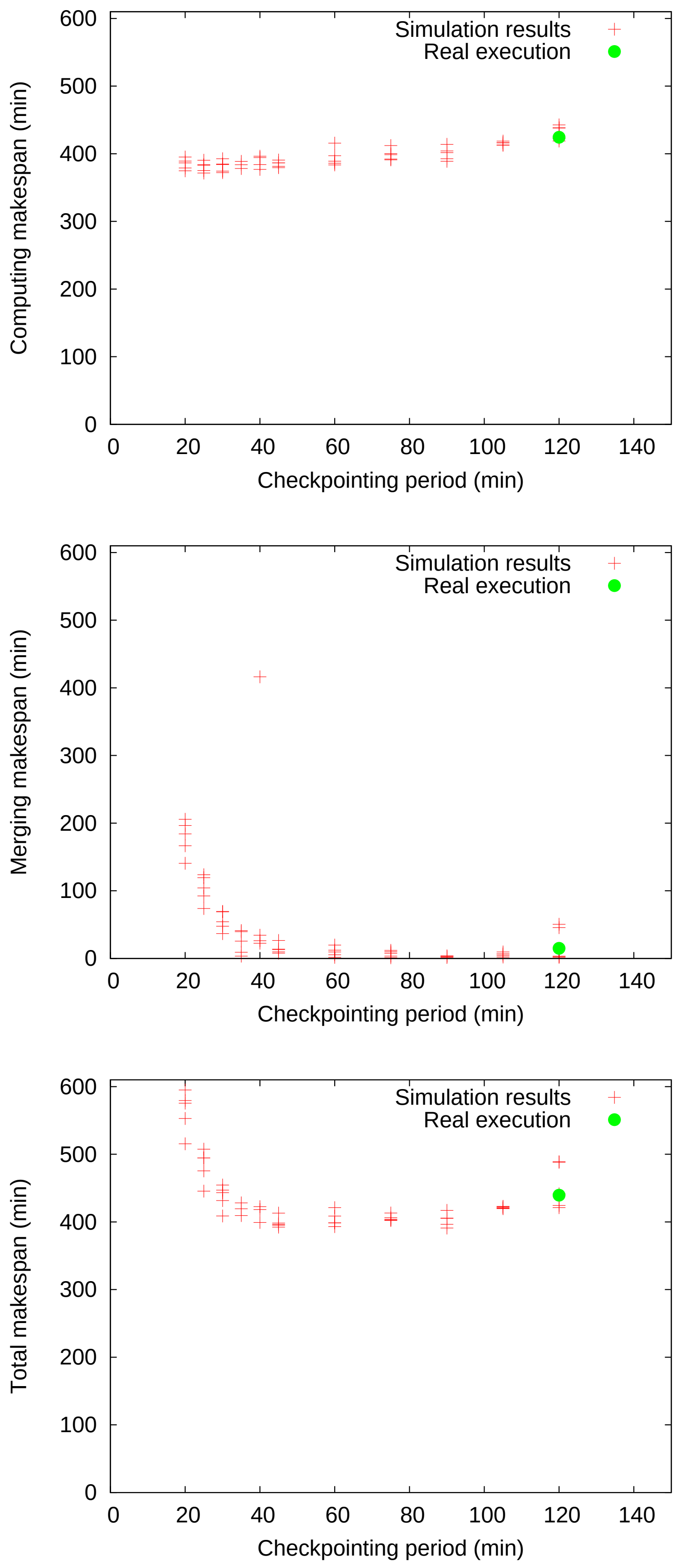


FIGURE A.5: Computing, merging and total makespans as a function of the checkpointing period.

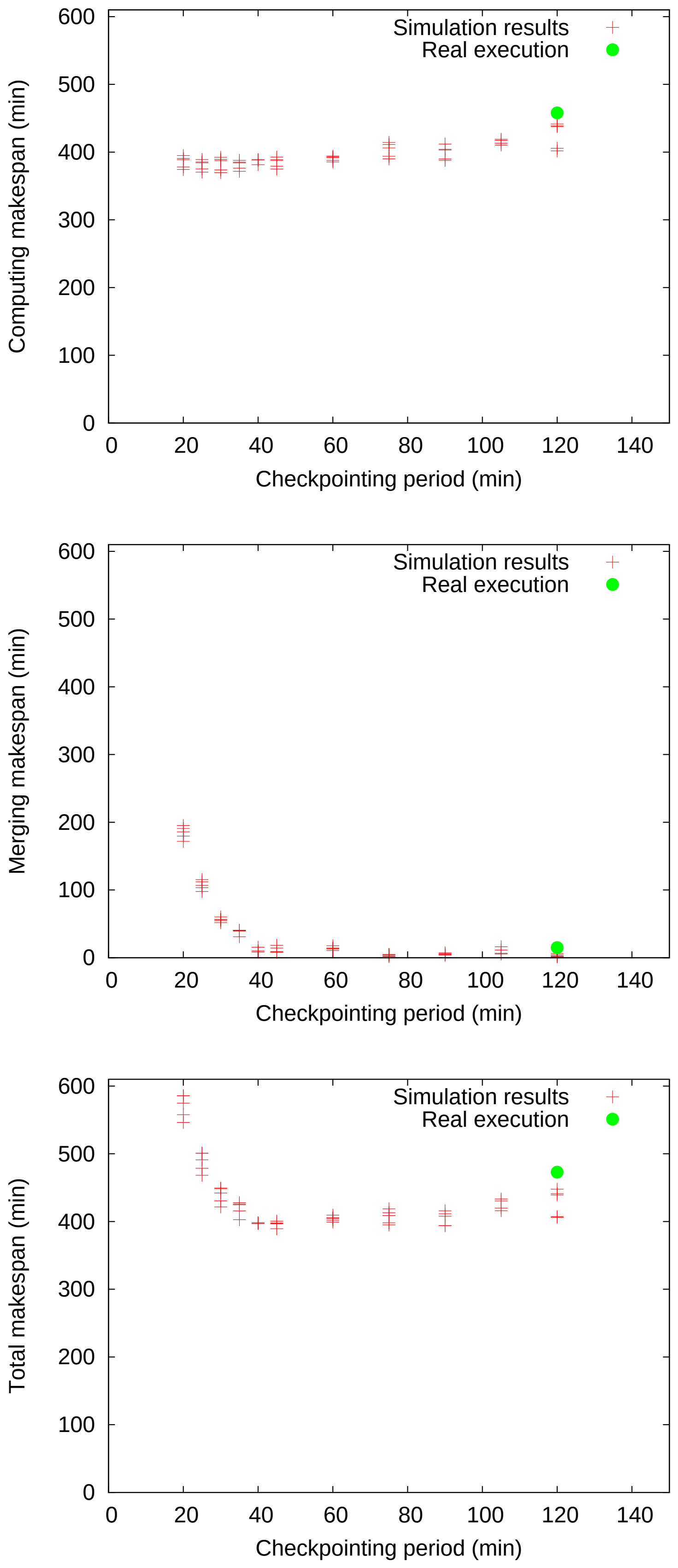


FIGURE A.6: Computing, merging and total makespans as a function of the checkpointing period.

Bibliographie

[Ahn et al., 2008] Ahn, S., Namgyu, K., Seehoon, L., Soonwook, H., Dukyun, N., Koblitz, B., Breton, V., and Sangyong, H. (2008). Improvement of Task Retrieval Performance Using AMGA in a Large-Scale Virtual Screening. In *NCM'08*, pages 456–463.

[Allison et al., 2006] Allison, J., Amako, K., Apostolakis, J., Araujo, H., Dubois, P., Asai, M., Barrand, G., Capra, R., Chauvie, S., Chytracek, R., Cirrone, G., Cooperman, G., Cosmo, G., Cuttone, G., Daquino, G., Donszelmann, M., Dressel, M., Folger, G., Foppiano, F., Generowicz, J., Grichine, V., Guatelli, S., Gumplinger, P., Heikkinen, A., Hrivnacova, I., Howard, A., Incerti, S., Ivanchenko, V., Johnson, T., Jones, F., Koi, T., Kokoulin, R., Kossov, M., Kurashige, H., Lara, V., Larsson, S., Lei, F., Link, O., Longo, F., Maire, M., Mantero, A., Mascialino, B., Mclaren, I., Lorenzo, P., Minamimoto, K., Murakami, K., Nieminen, P., Pandola, L., Parlati, S., Peralta, L., Perl, J., Pfeiffer, A., Pia, M., Ribon, A., Rodrigues, P., Russo, G., Sadilov, S., Santin, G., Sasaki, T., Smith, D., Starkov, N., Tanaka, S., Tcherniaev, E., Tome, B., Trindade, A., Truscott, P., Urban, L., Verderi, M., Walkden, A., Wellisch, J., Williams, D., Wright, D., and Yoshida, H. (2006). Geant4 Developments and Applications. *IEEE Transactions on Nuclear Science*, 53 :270–278.

[Altintas et al., 2004] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler : an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424.

[Anderson et al., 2002] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home : an experiment in public-resource computing. *Commun. ACM*, 45(11) :56–61.

[Antoniu et al., 2012] Antoniu, G., Brasche, G., Canon, S., and Fox, G. (2012). Science clouds experiences : sunny, cloudy or rainy ? In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 61–62, New York, NY, USA. ACM.

[Bal et al., 2000] Bal, H., Bhoedjang, R., Hofman, R., Jacobs, C., Kielmann, T., Maassen, J., van Nieuwpoort, R., Romein, J., Renambot, L., Rühl, T., Veldema, R., Verstoep, K., Baggio, A., Ballintijn, G., Kuz, I., Pierre, G., van Steen, M., Tanenbaum, A., Doornbos, G., Germans, D., Spoelder, H., Baerends, E.-J., van Gisbergen, S., Afsermanesh, H., van Albada, D., Belloum, A., Dubbeldam, D., Hendrikse, Z., Hertzberger, B., Hoekstra, A., Iskra, K., Kandhai, D., Koelma, D., van der Linden, F., Overeinder, B., Sloot, P., Spinnato, P., Epema, D., van Gemund, A., Jonker, P., Radulescu, A., van Reeuwijk, C., Sips, H., Knijnenburg, P., Lew, M., Sluiter, F., Wolters, L., Blom, H., de Laat, C., and van der Steen, A. (2000). The distributed asci supercomputer project. *SIGOPS Oper. Syst. Rev.*, 34(4) :76–96.

[Balaton et al., 2008] Balaton, Z., Farkas, Z., Gombas, G., Kacsuk, P., Lovas, R., Marosi, A., Terstyanszky, G., Kiss, T., Lodygensky, O., Fedak, G., Emmen, A., Kelley, I., Taylor, I., Cardenas-Montes, M., and Araujo, F. (2008). Edges : The common boundary between service and desktop grids. In Gorlatch, S., Fragopoulou, P., and Priol, T., editors, *Grid Computing*, pages 37–48. Springer US.

[Barbera et al., 2011] Barbera, R., Brasileiro, F., Bruno, R., Ciuffo, L., and Scardaci, D. (2011). Supporting e-science applications on e-infrastructures : Some use cases from latin america. In Preve, N. P., editor, *Grid Computing*, Computer Communications and Networks, pages 33–55. Springer London.

[Basney et al., 1999] Basney, J., Raman, R., and Livny, M. (1999). High throughput Monte-Carlo. In *PPSC*.

[Beaumont et al., 2011] Beaumont, O., Bobelin, L., Casanova, H., Clauss, P.-N., Donassolo, B., Eyraud-Dubois, L., Genaud, S., Hunold, S., Legrand, A., Quinson, M., Rosa, C., Schnorr, L., Stillwell, M., Suter, F., Thiery, C., Velho, P., Vincent, J.-M., and Won, Young, J. (2011). Towards Scalable, Accurate, and Usable Simulations of Distributed Applications and Systems. Rapport de recherche RR-7761, INRIA.

[Bell et al., 2003] Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., and Zini, F. (2003). Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications.*

[Ben-Yehuda et al., 2012] Ben-Yehuda, O., Schuster, A., Sharov, A., Silberstein, M., and Iosup, A. (2012). Expert : Pareto-efficient task replication on grids and a cloud. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 167–178.

[Bezzine et al., 2006] Bezzine, S., Galtier, V., Vialle, S., Baude, F., Bossy, M., Doan, V. D., and Henrio, L. (2006). A Fault Tolerant and Multi-Paradigm Grid Architecture for Time Constrained Problems. Application to Option Pricing in Finance. In *e-Science and Grid Computing.*

[Bobelin et al., 2012] Bobelin, L., Legrand, A., David, Alejandro González, M., Navarro, P., Quinson, M., Suter, F., and Thiery, C. (2012). Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation. In *CCGrid 2012 – The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, page 19, Ottawa, Canada. RR-7829 RR-7829.

[Bolze et al., 2006] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., et al. (2006). Grid'5000 : a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494.

[Bouguerra et al., 2011] Bouguerra, M., Kondo, D., and Trystram, D. (2011). On the scheduling of checkpoints in desktop grids. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 305 –313.

[Buyya et al., 2005] Buyya, R., Murshed, M., Abramson, D., and Venugopal, S. (2005). Scheduling parameter sweep applications on global grids : a deadline and budget constrained cost-time optimization algorithm. *Software : Practice and Experience*, 35(5) :491–512.

[Buyya and Murshed, 2002] Buyya, R. and Murshed, M. M. (2002). 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) :1175–1220.

[C. Noblet and Delpon, 2013] C. Noblet, S. Chiavassa, S. S. J. S. F. P. A. L. and Delpon, G. (2013). Simulation of the xrad225cx preclinical irradiator using gate/geant4. In *Workshop Radiobiology applied to Oncology.*

[Calheiros et al., 2011] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and Experience*, 41(1) :23–50.

[Camarasu-Pop et al., 2013a] Camarasu-Pop, S., Glatard, T., and Benoit-Cattin, H. (2013a). Simulating application workflows and services deployed on the european grid infrastructure. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 18–25.

[Camarasu-Pop et al., 2011] Camarasu-Pop, S., Glatard, T., Benoit-Cattin, H., and Sarrut, D. (2011). *Enabling Grids for GATE Monte-Carlo Radiation Therapy Simulations with the GATE-Lab.*

[Camarasu-Pop et al., 2013b] Camarasu-Pop, S., Glatard, T., da Silva, R. F., Gueth, P., Sarrut, D., and Benoit-Cattin, H. (2013b). Monte Carlo simulation on heterogeneous distributed systems : A computing framework with parallel merging and checkpointing strategies. *Future Generation Computer Systems*, 29(3) :728 – 738.

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

[Cappello and Bal, 2007] Cappello, F. and Bal, H. (2007). Toward an international "computer science grid". In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 3–12.

[Caron et al., 2007] Caron, E., Garonne, V., and Tsaregorodtsev, A. (2007). Definition, modelling and simulation of a grid computing scheduling system for high throughput computing. *Future Gener. Comput. Syst.*, 23(8) :968–976.

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

[Castro et al., 2004] Castro, R., Coates, M., Liang, G., Nowak, R., and Yu, B. (2004). Network tomography : Recent developments. *Statistical Science*, 19(3) :499–517.

[Chervenak et al., 2007] Chervenak, A., Deelman, E., Livny, M., Su, M.-H., Schuler, R., Bharathi, S., Mehta, G., and Vahi, K. (2007). Data placement for scientific applications in distributed environments. In *Grid Computing, 2007 8th IEEE/ACM International Conference on*, pages 267–274.

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

[Clauss et al., 2011] Clauss, P.-N., Stillwell, M., Genaud, S., Suter, F., Casanova, H., and Quinson, M. (2011). Single node on-line simulation of mpi applications with smpi. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 664 –675.

[Compostella et al., 2007] Compostella, G., Lucchesi, D., Griso, S. P., and Sfiligoi, I. (2007). CDF Monte Carlo Production on LCG Grid via LcgCAF Portal. In *E-SCIENCE '07 : Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 11–16, Washington, DC, USA. IEEE Computer Society.

[Condie et al., 2010] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., and Sears, R. (2010). MapReduce online. In *NSDI*, pages 313–328. USENIX Association.

[Crawl et al., 2011] Crawl, D., Wang, J., and Altintas, I. (2011). Provenance for mapreduce-based data-intensive workflows. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 21–30, New York, NY, USA. ACM.

[da Silva and Senger, 2011] da Silva, F. A. and Senger, H. (2011). Scalability limits of bag-of-tasks applications running on hierarchical platforms. *Journal of Parallel and Distributed Computing*, 71(6) :788 – 801.

[Dail and Desprez, 2006] Dail, H. and Desprez, F. (2006). Experiences with hierarchical request flow management for network-enabled server environments. *IJHPCA*, 20(1) :143–157.

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). MapReduce : simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA. USENIX Association.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). MapReduce : simplified data processing on large clusters. *Commun. ACM*, 51 :107–113.

[Deelman et al., 2005] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., and Katz, D. S. (2005). Pegasus : a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3) :219–237.

[Ekanayake and Pallickara, 2008] Ekanayake, J. and Pallickara, S. (2008). MapReduce for data intensive scientific analysis. In *Fourth IEEE International Conference on eScience*, pages 277–284.

[Elmroth and Gardfjall, 2005] Elmroth, E. and Gardfjall, P. (2005). Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In *Proceedings of the First International Conference on e-Science and Grid Computing*, E-SCIENCE '05, pages 221–229, Washington, DC, USA. IEEE Computer Society.

[Engh et al., 2003] Engh, D., Smallen, S., Gieraltowski, J., Fang, L., Gardner, R., Gannon, D., and Bramley, R. (2003). GRAPPA : Grid access portal for physics applications. *CoRR*, cs.DC/0306133.

[Epema, 2012] Epema, D. H. (2012). Twenty years of grid scheduling research and beyond (keynote talk). In *CCGrid'2011*, pages xxxi – xxxiii, Ottawa, CA.

[Fahringer et al., 2004] Fahringer, T., Pllana, S., and Villazon, A. (2004). A-gwl : Abstract grid workflow language. In Bubak, M., Albada, G., Sloot, P., and Dongarra, J., editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 42–49. Springer Berlin Heidelberg.

[Fahringer et al., 2007] Fahringer, T., Prodan, R., Duan, R., Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., and Wieczorek, M. (2007). ASKALON : A Development and Grid Computing Environment for Scientific Workflows. In Taylor, I., Deelman, E., Gannon, D., and Shields, M., editors, *Workflows for e-Science*, pages 450–471. Springer London.

[Fanfani et al., 2010] Fanfani, A., Afaq, A., Sanches, J., Andreeva, J., Bagliesi, G., Bauerdick, L., Belforte, S., Bittencourt Sampaio, P., Bloom, K., Blumenfeld, B., Bonacorsi, D., Brew, C., Calloni, M., Cesini, D., Cinquilli, M., Codispoti, G., DHondt, J., Dong, L., Dongiovanni, D., Donvito, G., Dykstra, D., Edelmann, E., Egeland, R., Elmer, P., Eulisse, G., Evans, D., Fanzago, F., Farina, F., Feichtinger, D., Fisk, I., Flix, J., Grandi, C., Guo, Y., Happonen, K., Hernandez, J., Huang, C.-H., Kang, K., Karavakis, E., Kasemann, M., Kavka, C., Khan, A., Kim, B., Klem, J., Koivumaki, J., Kress, T., Kreuzer, P., Kurca, T., Kuznetsov, V., Lacaprara, S., Lassila-Perini, K., Letts, J., LindÃ©n, T., Lueking, L., Maes, J., Magini, N., Maier, G., Mcbride, P., Metson, S., Miccio, V., Padhi, S., Pi, H., Riahi, H., Riley, D., Rossman, P., Saiz, P., Sartirana, A., SciabÃ , A., Sekhri, V., Spiga, D., Tuura, L., Vaandering, E., Vanelderen, L., Mulders, P., Vedaee, A., Villella, I., Wicklund, E., Wildish, T., Wissing, C., and Wurthwein, F. (2010). Distributed Analysis in CMS. *Journal of Grid Computing*, 8(2) :159–179.

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

[Ferreira da Silva and Glatard, 2012] Ferreira da Silva, R. and Glatard, T. (2012). A Science-Gateway Workload Archive to Study Pilot Jobs, User Activity, Bag of Tasks, Task Sub-Steps, and Workflow Executions. In *CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing*, Rhodes, GR.

[Ferreira da Silva et al., 2013] Ferreira da Silva, R., Glatard, T., and Desprez, F. (2013). Self-healing of workflow activity incidents on distributed computing infrastructures. *Future Generation Computer Systems*.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid : Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3) :200–222.

[Frincu et al., 2008] Frincu, M.-E., Quinson, M., and Suter, F. (2008). Handling Very Large Platforms with the New SimGrid Platform Description Formalism. Rapport Technique RT-0348, INRIA.

[Frisoni et al., 2011] Frisoni, G. B., Redolfi, A., Manset, D., Rousseau, M.-E., Toga, A., and Evans, A. C. (2011). Virtual imaging laboratories for marker discovery in neurodegenerative diseases. *Nature Reviews Neurology*, 7(8) :429–438.

[Galis et al., 2011] Galis, A., Clayman, S., Lefevre, L., Fischer, A., de Meer, H., Rubio-Loyola, J., Serrat, J., and Davy, S. (2011). Towards in-network clouds in future internet. In *The Future Internet - Future Internet Assembly 2011 : Achievements and*

Technological Promises, volume 6656, pages 19–33. Lecture Notes in Computer Science, Springer. ISBN 978-3-642-20897-3.

[Galyuk et al., 2002] Galyuk, Y. P., Memnonov, V., Zhuravleva, S. E., and Zolotarev, V. I. (2002). Grid technology with dynamic load balancing for Monte Carlo simulations. In *PARA '02 : Proceedings of the 6th International Conference on Applied Parallel Computing Advanced Scientific Computing*, pages 515–520, London, UK. Springer-Verlag.

[Garg and Singh, 2011] Garg, R. and Singh, A. (2011). Fault Tolerance in Grid Computing : State of the Art and Open Issues. *International Journal of Computer Science & Engineering Survey (IJCSES)*, 2(1).

[Germain Renaud et al., 2008] Germain Renaud, C., Loomis, C., Moscicki, J., and Texier, R. (2008). Scheduling for Responsive Grids. *Journal of Grid Computing*, 6 :15–27.

[Glatard and Camarasu-Pop, 2009] Glatard, T. and Camarasu-Pop, S. (2009). Modelling pilot-job applications on production grids. In *7th international workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (Heteropar 09)*, Delft, The Netherlands.

[Glatard et al., 2012] Glatard, T., Lartizien, C., Gibaud, B., Ferreira da Silva, R., Forestier, G., Cervenansky, F., Alessandrini, M., Benoit-Cattin, H., Bernard, O., Camarasu-Pop, S., Cerezo, N., Clarysse, P., Gaignard, A., Hugonnard, P., Liebgott, H., Marache, S., Marion, A., Montagnat, J., Tabary, J., and Friboulet, D. (2012). A virtual imaging platform for multi-modality medical image simulation. *IEEE Transactions on Medical Imaging*, in press.

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

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

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

[Gueth et al., 2013] Gueth, P., Dauvergne, D., Freud, N., LÃ©tang, J. M., Ray, C., Testa, E., and Sarrut, D. (2013). Machine learning-based patient specific prompt-gamma dose monitoring in proton therapy. *Physics in Medicine and Biology*, 58(13) :4563.

[Gustedt et al., 2009] Gustedt, J., Jeannot, E., and Quinson, M. (2009). Experimental validation in large-scale systems : a survey of methodologies. *Parallel Processing Letters*, 19(3) :399–418.

[Halton, 1970] Halton, J. H. (1970). A Retrospective and Prospective Survey of the Monte Carlo Method. *SIAM Review*, 12(1) :1–63.

[Iosup et al., 2008] Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., and Epema, D. H. J. (2008). The grid workloads archive. *Future Gener. Comput. Syst.*, 24(7) :672–686.

[Jacq et al., 2008] Jacq, N., Salzeman, J., Jacq, F., Legré, Y., Medernach, E., Montagnat, J., Maass, J., Reichstadt, M., Schwichtenberg, H., Sridhar, M., Kasam, V., Zimmermann, M., Hofmann, M., and Breton, V. (2008). Grid-enabled Virtual Screening against malaria. *Journal of Grid Computing (JGC)*, 6(1) :29–43.

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

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

[Janjic and Hammond, 2010] Janjic, V. and Hammond, K. (2010). Granularity-aware work-stealing for computationally-uniform grids. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 123–134.

[Josï¿½ Carlos Mouriï¿½o Gallego, 2007] Josï¿½ Carlos Mourï¿½o Gallego, Andrï¿½s Gï¿½mez, C. F. S. F. J. G. C. D. A. R. S. J. P. G. F. G. R. D. G. C. M. P. C. (2007). In *1st Iberian Grid Infrastructure Conference Proceedings (IBERGRID)*.

[Kacsuk, 2011] Kacsuk, P. (2011). P-GRADE portal family for grid infrastructures. *Concurr. Comput. : Pract. Exper.*, 23(3) :235–245.

[Kacsuk et al., 2008] Kacsuk, P., Farkas, Z., and Fedak, G. (2008). Towards making BOINC and EGEE interoperable. In *4th eScience Conference*, pages 478–484, Indianapolis.

[Kacsuk and Sipos, 2005] Kacsuk, P. and Sipos, G. (2005). Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. *Journal of Grid Computing (JGC)*, 3(3-4) :221 – 238.

[Krauter et al., 2002] Krauter, K., Buyya, R., and Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2) :135–164.

[Lassnig et al., 2010] Lassnig, M., Fahringer, T., Garonne, V., Molfetas, A., and Branco, M. (2010). Identification, Modelling and Prediction of Non-periodic Bursts in Workloads. In *CCGRID*, pages 485–494.

[Laure et al., 2006] Laure, E., Fisher, S., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Byrom, R., Cornwall, L., Craig, M., Meglio, A. D., Djaoui, A., Giacomini, F., Hahkala, J., Hemmer, F., Hicks, S., Edlund, A., Maraschini, A., Middleton, R., Sgaravatto, M., Steenbakkers, M., Walk, J., and Wilson, A. (2006). Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1) :33–45.

[Legrand et al., 2003] Legrand, A., Marchal, L., and Casanova, H. (2003). Scheduling distributed applications : the simgrid simulation framework. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 138 – 145.

[Li et al., 2010] Li, T., Camarasu-Pop, S., Glatard, T., Grenier, T., and Benoit-Cattin, H. (2010). Optimization of mean-shift scale parameters on the egee grid. In *Studies in health technology and informatics, Proceedings of Healthgrid 2010*, volume 159, pages 203–214.

[Lingrand et al., 2010] Lingrand, D., Montagnat, J., Martyniak, J., and Colling, D. (2010). Optimization of jobs submission on the EGEE production grid : modeling faults using workload. *Journal of Grid Computing (JOGC) Special issue on EGEE*, 8(2) :305–321.

[Litke et al., 2007] Litke, A., Skoutas, D., Tserpes, K., and Varvarigou, T. (2007). Efficient task replication and management for adaptive fault tolerance in mobile grid environments. *Future Generation Computer Systems*, 23(2) :163 – 178.

[Lu and Dinda, 2003] Lu, D. and Dinda, P. A. (2003). Synthesizing realistic computational grids. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 16–, New York, NY, USA. ACM.

[Ma et al.,] Ma, J., Liu, W., and Glatard, T. A classification of file placement and replication methods on grids. *to appear in Future Generation of Computer Science.*

[Maeno, 2008] Maeno, T. (2008). Panda : distributed production and distributed analysis system for atlas. *Journal of Physics : Conference Series*, 119(6) :062036 (4pp).

[Maeno et al., 2011] Maeno, T., De, K., Wenaus, T., Nilsson, P., Stewart, G. A., Walker, R., Stradling, A., Caballero, J., Potekhin, M., Smith, D., and Collaboration, T. A. (2011). Overview of atlas panda workload management. *Journal of Physics : Conference Series*, 331(7) :072024.

[Maheshwari et al., 2009] Maheshwari, K., Missier, P., Goble, C., and Montagnat, J. (2009). Medical Image Processing Workflow Support on the EGEE Grid with Taverna. In *Intl Symposium on Computer Based Medical Systems(CBMS'09)*. IEEE.

[Maigne et al., 2004] Maigne, L., Hill, D., Calvat, P., Breton, V., Lazaro, D., Reuillon, R., Legré, Y., and Donnarieix, D. (2004). Parallelization of Monte-Carlo simulations and submission to a grid environment. In *Parallel Processing Letters HealthGRID 2004*, volume 14, pages 177–196, Clermont-Ferrand France.

[Mairi and Nicholson, 2006] Mairi, C. and Nicholson, M. (2006). File management for hep data grids. Technical report.

[Marion et al., 2011] Marion, A., Forestier, G., Benoit-Cattin, H., Camarasu-Pop, S., Clarysse, P., da Silva, R. F., Gibaud, B., Glatard, T., Hugonnard, P., Lartizien, C., Liebgott, H., Tabary, J., Valette, S., and Friboulet, D. (2011). Multi-modality medical image simulation of biological models with the Virtual Imaging Platform (VIP). In *IEEE CBMS 2011*, Bristol, UK. submitted.

[Mascagni and Li, 2003a] Mascagni, M. and Li, Y. (2003a). Computational infrastructure for parallel, distributed, and grid-based Monte-Carlo computations. In *LSSC*, pages 39–52.

[Mascagni and Li, 2003b] Mascagni, M. and Li, Y. (2003b). Computational infrastructure for parallel, distributed, and grid-based Monte-Carlo computations. In *LSSC*, pages 39–52.

[M.G. Bugeiro, 2009] M.G. Bugeiro, J.C. Mouriï¿½o, A. G. C. V. E. H. I. L. y. D. R. (2009). Integration of slas with gridway in beineimrt project. In *3rd Iberian Grid Infrastructure Conference*.

[Montagnat et al., 2009] Montagnat, J., Isnard, B., Glatard, T., Maheshwari, K., and Blay-Fornarino, M. (2009). A data-driven workflow language for grids based on array programming principles. In *Workshop on Workflows in Support of Large-Scale Science(WORKS'09)*.

[Montresor and Jelasity, 2009] Montresor, A. and Jelasity, M. (2009). Peersim : A scalable p2p simulator. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 99 –100.

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

[Mościcki, 2003] Mościcki, J. T. (2003). Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research A*, 502 :426ï¿½429.

[Naicken et al., 2006] Naicken, S., Basu, A., Livingston, B., and Rodhetbhai, S. (2006). A survey of peer-to-peer network simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium.*

[Oinn et al., 2004] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., and Li, P. (2004). Taverna : A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20) :3045–3054.

[Pautasso and Alonso, 2006] Pautasso, C. and Alonso, G. (2006). Parallel computing patterns for grid workflows. In *Workflows in Support of Large-Scale Science, 2006. WORKS '06. Workshop on*, pages 1–10.

[Pena et al., 2009] Pena, J., Gonzalez-Castaï¿½o, D. M., Gomez, F., Gago-Arias, A., Gonzï¿½lez-Castaï¿½o, F. J., Rodrï¿½guez-Silva, D. A., Gonzï¿½lez, D., Pombar, M., Sï¿½nchez, M., Portas, B. C., Gï¿½mez, A., and Mouriï¿½o, C. (2009). E-IMRT : a web platform for the verification and optimization of radiation treatment plans. In Magjarevic, R., Nagel, J. H., Dï¿½ssel, O., and Schlegel, W. C., editors, *World Congress on Medical Physics and Biomedical Engineering, September 7 - 12, 2009, Munich, Germany*, volume 25/1 of *IFMBE Proceedings*, pages 511–514. Springer Berlin Heidelberg.

[Plank, 1997] Plank, J. (1997). An overview of checkpointing in uniprocessor and distributed systems, focusing on implementation and performance. Technical report.

[Procassini et al., 2005] Procassini, R., O'Brien, M., and Taylor, J. (2005). 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.

[Quinson et al., 2010] Quinson, M., Bobelin, L., and Suter, F. (2010). Synthesizing generic experimental environments for simulation. *International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing*, pages 222–229.

[Quintin and Wagner, 2012] Quintin, J. and Wagner, F. (2012). WSCOM : Online Task Scheduling with Data Transfers. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 344–351.

[Riteau et al., 2010] Riteau, P., Tsugawa, M., Matsunaga, A., Fortes, J., Freeman, T., Labissoniere, D., and Keahey, K. (2010). Sky Computing on FutureGrid and Grid'5000. In *TeraGrid'10.*

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

[Sadashiv and Kumar, 2011] Sadashiv, N. and Kumar, S. (2011). Cluster, grid and cloud computing : A detailed comparison. In *Computer Science Education (ICCSE), 2011 6th International Conference on*, pages 477–482.

[Samak et al., 2013] Samak, T., Morin, C., and Bailey, H., D. (2013). Energy consumption models and predictions for large-scale systems. In de Supinski, B. R. and Li, D., editors, *The Ninth Workshop on High-Performance, Power-Aware Computing*, Boston, United States. IEEE, IEEE.

[Santos-Neto et al., 2005] Santos-Neto, E., Cirne, W., Brasileiro, F., and Lima, A. (2005). Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *Job Scheduling Strategies for Parallel Processing*, volume 3277, pages 210–232.

[Sfiligoi, 2008] Sfiligoi, I. (2008). glideinWMS - A generic pilot-based workload management system. *Journal of Physics : Conference Series*, 119(6) :062044 (9pp).

[Stokes-Ress et al., 2009] Stokes-Ress, I., Baude, F., Doan, V.-D., and Bossy, M. (2009). Managing Parallel and Distributed Monte Carlo Simulations for Computational Finance in a Grid Environment. In Lin, S. and Yen, E., editors, *Grid Computing*, pages 183–204. Springer US.

[Sulistio et al., 2008] Sulistio, A., Cibej, U., Venugopal, S., Robic, B., and Buyya, R. (2008). A toolkit for modelling and simulating data grids : an extension to gridsim. *Concurrency and Computation : Practice and Experience*, 20(13) :1591–1609.

[Talia et al., 2008] Talia, D., Yahyapour, R., and Ziegler, W. (2008). *Grid Middleware and Services : Challenges and Solutions*. Springer Publishing Company, Incorporated, 1 edition.

[Tang and Fedak, 2012] Tang, B. and Fedak, G. (2012). Analysis of data reliability tradeoffs in hybrid distributed storage systems. In *The 17th IEEE International Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 2012), held in conjunction with IPDPS 2012, May 21-25, Shanghai, China.*, pages 1540–1549. IEEE Computer Society.

[Tannenbaum et al., 2001] Tannenbaum, T., Wright, D., Miller, K., and Livny, M. (2001). Condor – a distributed job scheduler. In Sterling, T., editor, *Beowulf Cluster Computing with Linux*. MIT Press.

[Taylor et al., 2007] Taylor, I., Shields, M., Wang, I., and Harrison, A. (2007). The Triana Workflow Environment : Architecture and Applications. In Taylor, I., Deelman, E., Gannon, D., and Shields, M., editors, *Workflows for e-Science*, pages 320–339. Springer London.

[Thain et al., 2005] Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice : The condor experience. *Concurrency and Computation : Practice and Experience*, 17 :2–4.

[Tsaregorodtsev et al., 2008] Tsaregorodtsev, A., Bargiotti, M., Brook, N., Ramo, A. C., Castellani, G., Charpentier, P., Cioffi, C., Closier, J., Diaz, R. G., Kuznetsov, G., Li,

Y. Y., Nandakumar, R., Paterson, S., Santinelli, R., Smith, A. C., Miguelez, M. S., and Jimenez, S. G. (2008). Dirac : a community grid solution. *Journal of Physics : Conference Series*, 119(6).

[Tsaregorodtsev et al., 2009] Tsaregorodtsev, A., Brook, N., Ramo, A. C., Charpentier, P., Closier, J., Cowan, G., Diaz, R. G., Lanciotti, E., Mathe, Z., Nandakumar, R., Paterson, S., Romanovsky, V., Santinelli, R., Sapunov, M., Smith, A. C., Miguelez, M. S., and Zhelezov, A. (2009). DIRAC3 . The New Generation of the LHCb Grid Software. *Journal of Physics : Conference Series*, 219 062029(6).

[Vu and Huet, 2013] Vu, T.-T. and Huet, F. (2013). A lightweight continuous jobs mechanism for mapreduce frameworks. In *13th IEEE/ACM International Symposium on Cluster, Grid and Cloud Computing (CCGRID 2013)*, page To Appear.

[Wang et al., 2009] Wang, G., Butt, A., Pandey, P., and Gupta, K. (2009). A simulation approach to evaluating design decisions in mapreduce setups. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pages 1 –11.

[Wang et al., 2012a] Wang, L., Camarasu-Pop, S., Tristan, G., Zhu, Y.-M., and Magnin, Isabelle, E. (2012a). Diffusion MRI Simulation with the Virtual Imaging Platform. In *journées scientifiques mésocentres et France Grilles 2012*, Paris, France.

[Wang et al., 2012b] Wang, L., Zhu, Y., Li, H., Liu, W., and Magnin, I. E. (2012b). Multiscale modeling and simulation of the cardiac fiber architecture for dmri. *IEEE Trans. Biomed. Engineering*, 59(1) :16–19.