Noname manuscript No.
(will be inserted by the editor)

Dynamic partitioning of GATE Monte-Carlo simulations on EGEE

Sorina Camarasu-Pop · Tristan Glatard · Jakub T. Mościcki · Hugues Benoit-Cattin · David Sarrut

Received: date / Accepted: date

Abstract The EGEE grid offers the necessary infrastructure and resources for reducing the running time of particle tracking Monte-Carlo applications like GATE. However, efforts are required to achieve reliable and efficient execution and to provide execution frameworks to end-users. This paper presents results obtained with porting the GATE software on the EGEE grid, our ultimate goal being to provide reliable, user-friendly and fast execution of GATE to radiation therapy researchers. To address these requirements, we propose a new parallelization scheme based on a dynamic partitioning and its implementation in two different frameworks using pilots jobs and workflows. Results show that pilot jobs bring strong improvement w.r.t. regular gLite submission, that the proposed dynamic partitioning algorithm further reduces execution time by a factor of two and that the genericity and user-friendliness offered by the workflow implementation do not introduce significant overhead.

Keywords GATE · Monte-Carlo simulations · grid computing · EGEE · dynamic particle parallelism · workflows · pilot jobs

1 Introduction

The OpenGate collaboration has been developing for many years an open-source software named GATE to perform nuclear medicine simulations, especially for TEP and SPECT imaging [10]. A new GATE module[1] focusing on radiation therapy simulations is currently being developed. Based on the Monte-Carlo toolkit Geant4 [2], GATE is a

S. Camarasu-Pop, T. Glatard, H. Benoit-Cattin, D. Sarrut
CREATIS, CNRS, INSERM, University of Lyon
INSA LYON, Bat Blaise Pascal
7 Avenue Jean Capelle
69621 Villeurbanne Cedex
Tel.: +33-472437299
E-mail: sorina.pop@creatis.insa-lyon.fr

J. T. Mościcki
CERN, Geneva, Switzerland

[1] Publicly available in GATE v6.0 (beginning of 2010)

collaborative development gathering researchers from several international institutions and is used by hundreds of persons worldwide.

The simulation in a particle tracking Monte-Carlo system consists in the successive stochastic tracking through matter of a large set 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. The simulation can be analyzed when the number of simulated particles is large enough, i.e. when a desired statistical uncertainty has been reached. As the physical interactions can also produce other particles that must also be tracked, typical radiation therapy simulations can take hours or days to complete.

Among radiation therapy simulation methods, Monte-Carlo approaches like GATE are known to be the most accurate but they are heavy to use because of their prohibitive computing time. Reducing their computing time is therefore of great importance and production grid infrastructures are well adapted to this kind of simulations, both in terms of cost and efficiency. Different parallelization methods for Monte-Carlo simulations have been proposed for execution on distributed environments. However, they are mostly cluster-oriented and still need improvement for grid usage, as reviewed in Section 2.

The EGEE grid, with its 250 resource centers, is a good candidate for reducing the running time of applications like GATE. EGEE is currently the largest production grid worldwide providing more than 40,000 CPUs and several Petabytes of storage. This infrastructure is used on a daily basis by thousands of scientists organized in over 200 Virtual Organizations (VOs). EGEE is operated by the gLite [13] middleware, which provides high-level services for scheduling and running computational jobs, as well as for data and grid infrastructure management.

Such a wide infrastructure is naturally heterogeneous and gLite job submission has to be coupled with other tools in order (i) to ensure reliability and high performance and (ii) to facilitate application porting and to offer high-level execution interfaces [23]. One possible solution for (i) is offered by pilot jobs, while (ii) can be addressed by workflow engines. Pilot jobs have been introduced during the last years [19,1,8,11, 28,27,14,3] and are now extensively used to improve performance and cope with the latencies and recurrent errors caused by the grid heterogeneity. Much effort has also been put into workflow technology to facilitate application porting and reusability [12, 15,7,5,21]. Engines now allow the execution of workflow applications on various grid middleware in a generic way.

In this paper, we propose a new dynamic particle partitioning method for GATE on distributed platforms, allowing for dynamic load balancing and ensuring complete results in spite of the failures that may occur. We also present two different implementations. The first one is based on the DIANE [18] pilot-job system. The second one also uses DIANE but is integrated into the MOTEUR [7] workflow engine, offering a higher genericity and better delivery to end-users.

The paper is organized as follows. Section 2 discusses related work. It presents different methods of parallelisation of Monte-Carlo simulations on distributed infrastructures, their limitations in the context of a large heterogeneous infrastructure like the EGEE grid, as well as existing solutions to some of those limitations. Section 3 describes the algorithm and the two implementations proposed for the dynamic partitioning of GATE on EGEE. Experiments and results are presented in Section 4 and quantify the gain yielded by DIANE pilot jobs w.r.t regular gLite submission for GATE,

the gain of using dynamic parallelization w.r.t static and the overhead of using a workflow manager above DIANE pilot jobs. Section 5 concludes the paper.

2 Parallel Monte-Carlo on distributed infrastructures

Based on the Geant4 Monte-Carlo software, GATE simulations can require a high computing time, but are naturally parallelizable. Instead of sequentially simulating a large number (up to several billions) of particles that may take several days or even weeks to complete, smaller groups (bags) of particles can be simulated independently and eventually merged. This process is valid only if the sub-simulations are statistically independent, which is guaranteed by using the random number generator integrated in GATE [26].

Previous work has been done on the parallelization of Monte-Carlo simulations in a grid environment. In [16], Maigne et al. present results obtained by running GATE in parallel on multiple processors of the DataGrid[2] project (EGEE predecessor). This approach relies on a pre-installation of GATE on computing sites and then on the submission of grid jobs to these particular sites. In this case particle parallelism is used, i.e. the geometry information is replicated on each processor and particles are distributed equally between processors.

In [24], Procassini et al. use both particle and spatial parallelism for the load balancing of parallel Monte-Carlo transport simulations. 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. In [24], a dynamic load balancing algorithm distributes the available processors to the spatial domains. The particles are then divided evenly and once and for all between the processors allocated to the same domain.

In the two examples mentioned above the number of particles simulated on each of the N processors represents a fraction P/N of the total of P particles. However, this static even distribution of particles may underexploit resources if adopted on heterogeneous platforms like grids. Indeed, fastest resources would rapidly complete their tasks and then remain idle until the end of the simulation. Such a poor load balancing dramatically slows down the application, in particular when a task is allocated to a slow resource towards the end of the simulation [4].

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 the EGEE grid, where the success rate within the biomed VO has been noticed to be of the order of 80 to 85% [9]. 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 (therefore particle) replication [4]. One replication method for grid-based Monte-Carlo calculations is presented in [17]. It uses the "N out of M strategy", i.e. it increases the number of subtasks from N to M, where M>N. Thus M parallel computations are submitted to the distributed environment.

[2] http://eu-datagrid.web.cern.ch

As soon as N results are ready, the final result can be produced. The main drawback of this solution 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.

Consequently, statically partitioning Monte-Carlo simulations (i.e. assigning a fixed number of particles to each task at the beginning of the simulation) and replication have several limitations in terms of efficacy (the performance delivered to the application) and efficiency (the amount of resources wasted) on wide-scale heterogeneous grid infrastructures such as EGEE.

Another alternative is the dynamic distribution and/or reassignment of particles to available processors during runtime. Dynamic partitioning is proposed in [6] and in [24] for spatial parallelism. In [6], the parallelization is done using an MPI implementation and is based on a semaphore principle under distributed memory conditions. In [24], communications are generated between processors in order to transmit changes from the last state. These two implementations are therefore cluster oriented and are not adapted for grid usage where communications between processors are very costly.

Pilot jobs [19] are a computing framework that provides an intermediate solution, by proposing a dynamic distribution (and reassignment) of statically partitioned tasks. In this case, a simulation is statically divided into a large number of tasks (of a convenient granularity) that are dynamically distributed to pilots by the workload manager (called master in the DIANE pilot-job framework). Tasks are no longer pushed to the batch manager but are put in a master pool and pulled by pilots running on available resources (grid nodes). Although pilots are still submitted through the regular grid middleware, such a pull model has several advantages in terms of latency reduction and fault-tolerance [25]. Latency is reduced by taking advantage of the fastest pilots (little queuing time and/or powerful processor) which will pull a maximum of tasks from the master pool, whereas fault-tolerance is achieved by removing the faulty pilots and resubmitting failed tasks to a different pilot.

By providing late task binding to resources, pilot jobs are a way to dynamically balance the number of particles assigned to computing resources. However, task granularity remains important. A very fine granularity (e.g. one particle per task) is equivalent to a dynamic particle partitioning (each pilot would fetch and execute 1-particle 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 granularity (large number of particles per task, therefore small number of tasks comparable to available resources) reduces the overhead but becomes almost equivalent to a static approach, leading to poor scheduling in heterogeneous non-reliable environments such as EGEE. To illustrate this point, Figure 1 plots the task flow of a 16-hour simulation split into 75 tasks and executed on EGEE submitting 75 pilot jobs. Load balancing is clearly sub-optimal. 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. Since tasks all correspond to the same number of particles, heterogeneity has 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 task partitioning strategy that would allow to balance the number of particles that each resource has to simulate depending on its performance. Given the wide scale of the target grid infrastructure, communications between tasks, between tasks and the master and between tasks and output storage

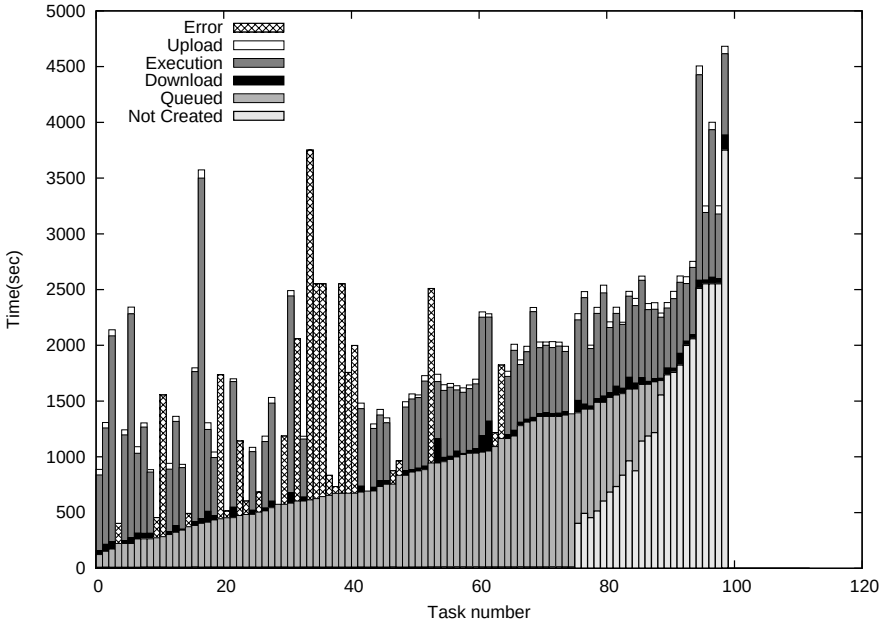


Fig. 1 Example of task flow obtained with static partitioning of a GATE simulation on EGEE and using pilot jobs. The simulation was split in 75 tasks but 24 of them failed for various reasons (4 data transfer issues, 6 pilots killed and 14 application errors). Hatched bars figure the time during which 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.

elements have to be avoided as much as possible. Moreover, task replication should not be widely employed given that EGEE is a shared infrastructure. Next section presents the algorithm and the two proposed implementations.

3 Algorithm and implementations

3.1 Dynamic task partitioning algorithm

The proposed dynamic task partitioning uses pilot jobs and consists in a "do-while" algorithm with no initial splitting. Each independent task of a simulation is created with the total number of particles and keeps on running until the desired number of particles is reached (with the contribution of all the tasks). Therefore, each tasks is liable to simulate the totality if the other tasks do not manage to simulate anything. The total number of simulated particles is given by the sum of all particles simulated by the independent tasks. Thus each computing resource contributes to the whole simulation until it is completed. Due to the statistical properties of GATE simulations, this does not require any communication between tasks. Only light communications are performed between tasks and the master.

Algorithm 1 Master algorithm for dynamic load balancing of GATE simulations

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

Algorithm 2 Pilot algorithm for dynamic load balancing of GATE simulations

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

It is to note that all the tasks have the same inputs but a different random seed number which will allow each task to simulate a unique set of particles complying with the simulation properties. Thus, if the different tasks of the same simulation are statistically independent, they generate independent sets of particles that can eventually be merged. Tasks are rendered statistically independent by assigning each of them a seed taken from a suite of non-correlated seed numbers. These seeds are generated using the random number generator as presented in Section 2.

Algorithm 1 shows the scheduling implemented by the master and algorithm 2 presents the pilot pseudo-code. The master periodically sums up the number of simulated particles 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. Outputs can be uploaded periodically (DIANE-only implementation, Section 3.2.1) or at the end of the simulation (workflow implementation, Section 3.2.2), only a single output data transfer per resource being thus required. Light communications between tasks and master occurs periodically to upload the current number of particles computed by the task and at the end of the simulation when the master sends stop signals.

Errors are nicely handled by this algorithm. When a task fails the remaining ones just keep on running until all the particles have been simulated. No tasks resubmission is thus required.

3.2 Implementation

The GATE code had to be extended to handle stop signals during simulation. This add-on allows the application to pause regularly (at a frequency that can be specified in the configuration macro file), 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

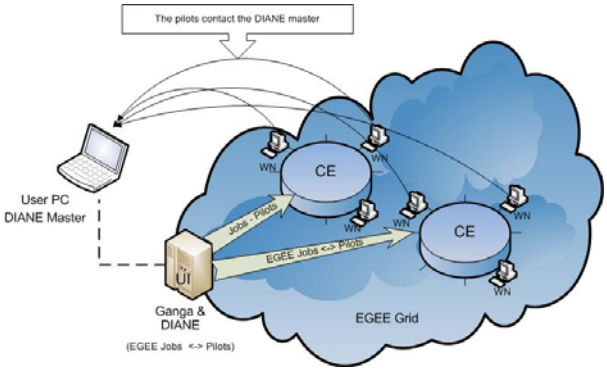


Fig. 2 DIANE Master Pilots architecture.

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. Note that this implementation may lead to computing slightly more particles than initially needed.

In the following sections, the implementation of the dynamic task partitioning in two different environments is presented. For comparison purposes both static and dynamic partitioning are presented. In all cases, on-the-fly download, installation and execution of GATE on the worker nodes is performed. The executable and the 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. This approach offers a significant flexibility allowing to change the software version at each submission and is not restricted to computing nodes on which the software is already installed.

3.2.1 Implementation in DIANE pilot-job framework

The implementation in the DIANE pilot-job framework consists in having DIANE masters and pilots dedicated to GATE simulations. DIANE pilots are submitted independently to the grid through the Ganga [20] frontend from a User Interface (UI) machine as shown in Figure 2. In our case, the DIANE master is launched on the user's computer, where the executable and all input files are stored. Once launched, the master generates the GATE tasks and distributes them to the registered pilots. Pilots download the executable and the inputs from the master to the Worker Nodes and upload the results to the master.

In such a dedicated setup meant to be used by one user (on whose computer the master is installed) for a particular application (in our case GATE), no permanent storage on the grid is needed. Consequently, in this dedicated environment files are not stored on EGEE SEs and thus the transfer time is reduced.

In the static approach, pilots execute GATE tasks that are from the beginning assigned a fraction P/N from the total number of particles (where P is the total number of particles and N the total number of tasks created). Pilots upload their results at the end of the simulation. The master reassigns tasks if they fail and considers the simulation finished when all tasks are successfully completed.

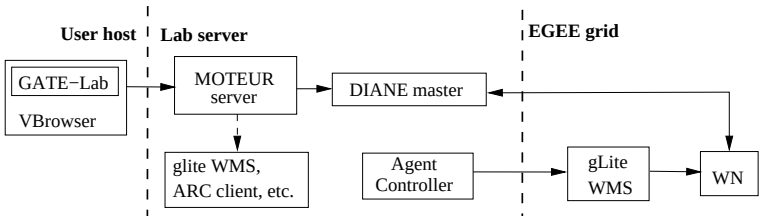


Fig. 3 Workflow environment. VBrowser and GATE-Lab plugins offer GUI for data management and experiment description. MOTEUR enacts application workflows and submit tasks to a DIANE master offering pilot-job execution.

The dynamic approach follows the algorithms presented in Algorithms 2 and 1. Each GATE task is initially assigned the total number of particles P. Nevertheless, it will probably (unless in the unlikely event that none of the other tasks get executed) be stopped before completing the P particles. In this case, knowing that one task may last a rather long time, results are uploaded on the master periodically (every 5 minutes) so that the risk of losing large tasks towards their end is reduced. This is possible because, as explained previously, in this dedicated implementation file transfer is direct between master and pilots, therefore faster and not cumbersome for the grid. The master has a counter for already simulated particles that is updated every time the pilots upload their current status. When the total number of particles is reached (the simulation is complete) the master finishes and the pilots die automatically having no master to connect to.

This integration in the DIANE pilot-job framework is quite straightforward and the results are encouraging as will be shown in Section 4.2. Nevertheless, it has certain limitations that can be addressed by integrating it into a workflow environment. First, this implementation is application-specific, i.e. the code of the pilots and the master is dedicated to one application. When using a workflow environment, the specificity can be handled at the workflow level. Generic pilots and master can thus be used. More generally, a middleware-independent workflow description of the application simplifies migration to other execution frameworks when pilot jobs are not the optimal solution. For instance, some resource providers may restrict network connectivity for security reasons or the target pilot-job system may not support some specific resources such as GPUs or MPI. Last but not least, delivery to end users may be difficult. Deployment of this solution requires individual handling of pilot submission and other technical actions (e.g. port opening for the DIANE master) that are not straightforward for all users. GATE execution on the grid should be provided as a service to radiation therapy researchers. Therefore, a higher level execution environment as described in Section 3.2.2 has been implemented. The next section presents our workflow implementation to address these requirements.

3.2.2 A workflow-based implementation

GATE has been integrated in the workflow system supporting grid applications in use at the Creatis laboratory[3]. It extends the system described in [23] to include DIANE. As illustrated in Figure 3, it complies to a 3-tier architecture composed of (i)

[3] http://www.creatis.insa-lyon.fr

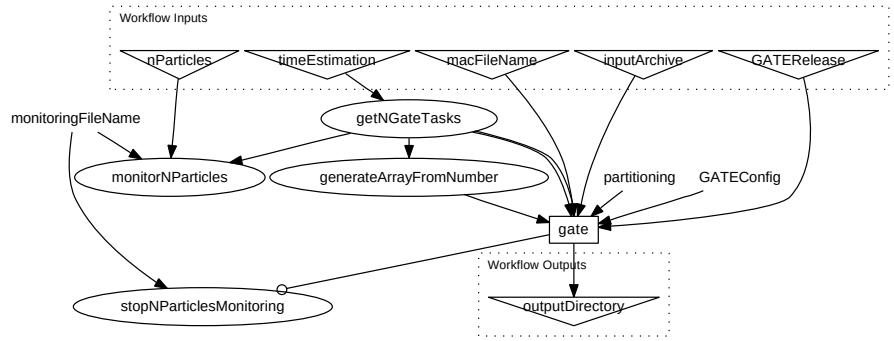


Fig. 4 Static GATE workflow. In/output files and parameters are figured by triangles. Ellipses denote locally executed components and the box figures a grid job. Labels with no shape are string constants. Arrows are data links and the circle-terminated line is a precedence constraint.

a client offering a GUI for grid data management and GATE simulation preparation (ii) a service managing DIANE master setup, pilot submission and task submission, monitoring and error handling and (iii) the grid itself, externally administrated and accessible through gLite.

The user interacts with a client made of the VBrowser (GUI to grid storage resources [22]) and a specific plugin (GATE-Lab) performing parameter checking, input files bundling and uploading, GATE release selection and history management. A "one-click" GATE simulation is made possible for the user who does not have to be aware of grid internals.

The server hosts DIANE and the MOTEUR workflow engine [7]. It has the gLite clients installed and ports opened to allow pilot connections. MOTEUR generates grid tasks from a Taverna ([21]) workflow description and submits them to a DIANE master using a generic task manager. DIANE pilots download and run tasks on worker nodes (WN), periodically uploading standard output and standard error to the master. Using this setup, any workflow run with MOTEUR is able to benefit from DIANE pilot jobs with no additional porting effort. Error handling is implemented by MOTEUR by resubmitting tasks up to a maximal number of times when they fail. To ensure basic security, user credentials are delegated to the MOTEUR server, which starts a new DIANE master for every user. Therefore jobs run with personal user credentials. Pilot jobs are submitted by an agent controller using Ganga, as for the DIANE-only implementation. Alternately, MOTEUR can submit tasks to other grid middleware such as gLite WMS (e.g. to run MPI jobs) or ARC client (e.g. to access NorduGrid resources).

All the files are stored on EGEE Storage Elements (SEs) and registered in the Logical File Catalog (LFC). Results are thus available permanently for post-processing (e.g. merging) and important data volumes can be stored with no maintenance costs for the application.

The static GATE workflow is shown in Figure 4. Component `gate` is the most important. The others are auxiliary components used for monitoring and simulation steering purposes. The corresponding workflow document in available online on the

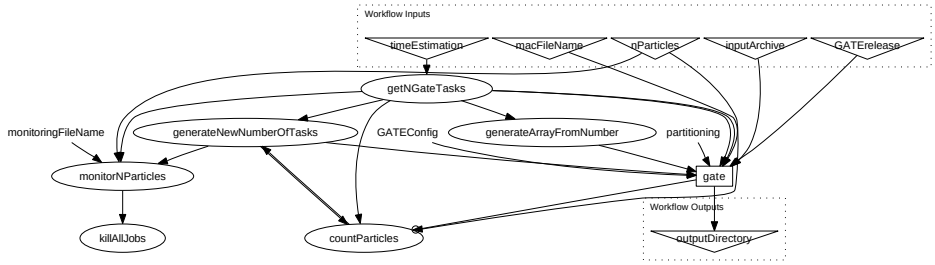


Fig. 5 Dynamic GATE workflow. Additional components have been added to the static workflow (Figure 4) to kill GATE tasks once the total number of particles to simulate is reached and to resubmit tasks in case of output data transfer errors.

myExperiment [4] website. It builds GATE tasks and submits them to the DIANE master. It takes as input the GATE input files (macro simulation description file and input archive), an archive containing the GATE release, the total number of tasks to simulate, the partitioning method (static or dynamic) and it iterates on an array containing all task numbers. Components `getNGateTasks` and `generateArrayFromNumber` are executed locally and determine the total number of GATE tasks to generate, given an estimation of the total workload (`timeEstimation`). Components `monitorNParticles` and `stopNParticlesMonitoring` also run locally. They produce files exploited by the GATE-Lab client to provide monitoring information to the user.

The dynamic workflow shown in Figure 5 only implements minor additions to the static. A component is added to kill gate tasks (i.e. send stop signals to running tasks and cancel scheduled ones) when the total number of particles has been simulated. Besides, once all tasks complete (i.e. they have received a stop signal and uploaded their results), the number of particles simulated is counted again and another set of tasks is generated if the desired number of particles is not reached. This is an extension to algorithm 1 meant to avoid particle loss due to output transfer errors.

Experiments conducted with both implementations (DIANE-only and workflow-based) for each of the static and dynamic approaches are presented in the next section.

4 Experiments and results

4.1 Conditions

Experiments reported in this section aim at identifying the gain provided by pilots and dynamic load balancing for GATE, as well as the overhead of using a workflow manager above DIANE pilot jobs. To address these questions several execution scenarios are considered (see Table 1). We tested standard static parallelization with gLite-only submission, static and dynamic parallelization within the DIANE-only framework, as well as static and dynamic parallelization within the workflow framework.

The first scenario implements static parallelization with standard WMS submission, i.e. job descriptions created with the JDL (Job Description Language) and submitted

[4] http://www.myexperiment.org/workflows/766

with the `glite-wms-job-submit` command. In this scenario the executable and input files are stored on one SE and downloaded by the running job on the WN. Outputs are uploaded on the same SE by each job at the end of the GATE execution.

The second and third scenarios use the DIANE-only implementation. The second corresponds to the static parallelization (DS) and the third to the dynamic (DD). In both cases the executable and all input files are stored on the same computer on which the DIANE master is running and where output results are also uploaded. The upload of results is done once at the end of each task for the static approach and regularly (once every 300s) during the execution of the simulation for the dynamic approach.

The forth and fifth scenarios use the workflow framework. The forth corresponds to static parallelization (WS) and the fifth to the dynamic (WD). In both cases, the executable and all input files are stored on one SE and downloaded by the running task on the WN. Outputs are uploaded on the same SE by each task at the end of the GATE execution for both approaches. Stdin/stdout files (of a size of a few KB only) are uploaded/downloaded every minute between the WN and the DIANE master for monitoring.

Scenario	Execution mode	Splitting approach	File storage	Resubmission
1: gLite	Standard WMS	static	SE	No
2: DS	DIANE only	static	Local	Tasks only
3: DD	DIANE only	dynamic	Local	No
4: WS	Workflow + DIANE	static	SE	Tasks only
5: WD	Workflow + DIANE	dynamic	SE	No*

Table 1 Experiment scenarios. *Tasks ressubmission only in case of output transfer errors (see Section 3.2.2)

Each experiment consists in executing a 16-hour simulation of 450,000 particles. All simulations are split in turn into 25, 50 and 75 tasks. For comparison reasons (between the gLite and the pilot-job frameworks) the number of submitted pilots for each experiment corresponds to the number of tasks created for that experiment. For the three scenarios that implement the static approach (gLite, DS, WS) simulations run with 25 jobs/pilots have tasks approximately three times longer (average of 40 min on EGEE resources) than those run with 75 jobs/pilots (average of 14 min on EGEE resources). Note that for the two dynamic approaches (DD and WD), each task is initially assigned the total of P particles. Therefore the performance is only influenced by the number of registered agents and the performance of the machines on which they run.

We would like to draw the reader's attention to the distinction between what we call a job and a task. The notion of a job in this context is associated to a gLite submission, i.e. the total amount of work that can be executed by a standard gLite job or a DIANE pilot (that has been submitted on the grid as a gLite job). On the contrary, the notion of task refers to a partial amount of work that has been defined at the application level and is executed by DIANE pilots. It is to note that a pilot, which is associated to a single job, can execute more than one task during its lifetime. In the pilot-job model, the application is split into a given number of tasks that are progressively assigned to registered pilots. In this context, failed jobs (DIANE pilots or gLite jobs) are not resubmitted. On the contrary, for pilot-job implementations, 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. Therefore, as initially the number of tasks is equal to the number of submitted pilots, task resubmission does not make sense for the dynamic implementation. In realistic conditions job resubmission is essential for the classical gLite job submission 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 reasons.

In all cases a single WMS is used. No requirement is used for job submission, i.e., jobs are sent to the whole EGEE `biomed` VO. This is convenient in our case as the application has been packaged so that it does not have any specific requirements. However, this may not be the case for other applications.

To ensure that the scenarios compared here are run in similar grid conditions, experiments from scenarios 1, 2 and 4 are submitted simultaneously (batch 1) and experiments from scenarios 2, 3 and 5 (batch 2) as well. Scenario 2 (DS) was therefore repeated twice and we will refer to it as DS1 for the first batch and DS2 for the second one. Each scenario consists of 3 experiments, each with 25, 50 and 75 tasks respectively. Moreover, each experiment is repeated 3 times. In total, 54 GATE simulations are conducted (3 repetitions $\times$ 3 job numbers $\times$ 3 scenarios $\times$ 2 batches). In the following, each simulation has a unique ID of the form `Scenario.ExperimentRepetitionNb-NbOfTasks`. For example, WS.2-50 is the second repetition of the 50 task experiment of the WS scenario.

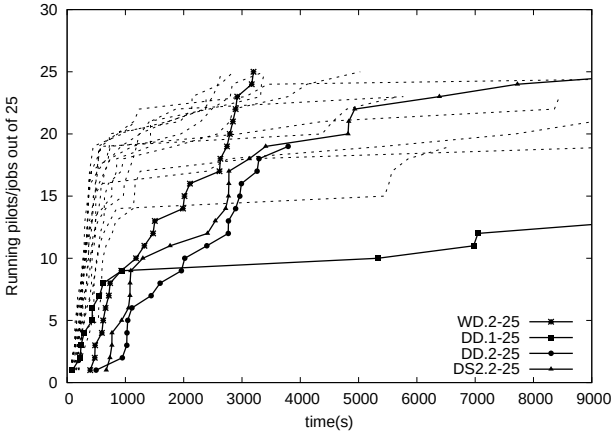
Data is dealt with using the gLite data management services. Data is stored on a grid SE and registered in the biomed logical file catalog (LFC). It can be uploaded or deleted using the standard gLite command-line applications or the VBrowser GUI. Space usage and quotas on the SE are managed by local site administrators according to the their policy. For comparison reasons, we use the same biomed SE for all experiments. This SE has been selected arbitrarily among a list of reliable biomed SEs.

Results presented in the next section aim at quantifying:

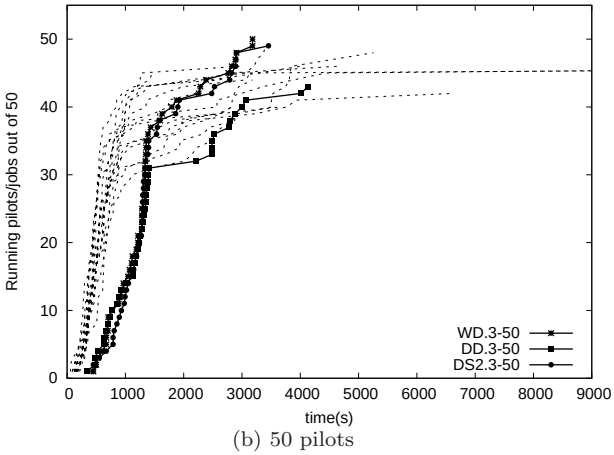
- The gain from using DIANE pilot jobs w.r.t regular gLite submission for GATE, as presented in 4.2.1
- The gain from using dynamic parallelization w.r.t static, as presented in 4.2.2
- The overhead of using a workflow manager above DIANE pilot jobs, as presented in 4.2.3

4.2 Results and discussion

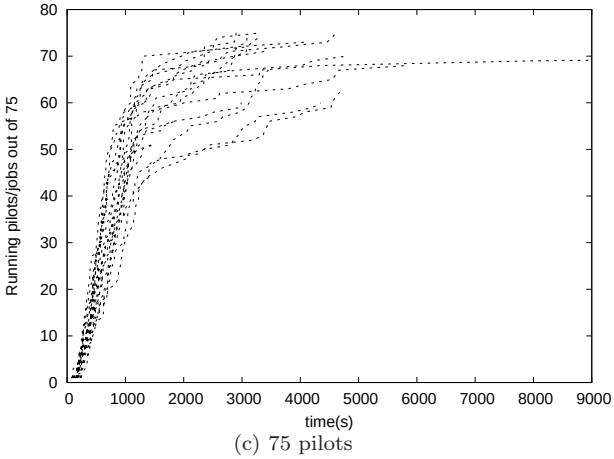
Figure 6 displays the number of registered DIANE pilots (for scenarios 2, 3, 4 and 5) and the number of running gLite jobs (for scenario 1) along time. Overall, it shows that no scenario was favored by the grid scheduler although there is some variability among repetitions. In figure 6(a) most of the runs have similar behavior, with about half of the submitted jobs/pilots registered in less than 15 minutes (900 seconds). We notice however a group of three singular runs: DS2.2-25, DD.2-25 and WD.2-25. They correspond to 3 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. DD.1-25 is clearly an outlier with very few registered pilots, which penalizes the overall performance of the experiment as will be discussed in section 4.2.2. In figure 6(b) there is another group of three singular runs corresponding to simultaneously executed experiments: DS2.3-50,



(a) 25 pilots



(b) 50 pilots



(c) 75 pilots

Fig. 6 Registered jobs/pilots. Dashed lines plot runs for which the evolution of the number of registered jobs/pilots is similar. Overall no scenario is favoured by the grid scheduler.

DD.3-50 and WD.3-50. In figure 6(c) the evolution of the number of registered jobs is overall very homogeneous among runs.

Figures 7, 8-(a,b,c), 10-(a,b,c) and 11-(a,b,c) plot the evolution of the simulation along time, i.e. the number of simulated particles out of the total of 450,000 particles. Each scenario is 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 particles respectively.

The application makespan (i.e. the time needed to complete 100% of the simulation) can be considered as a measure of the performance. However, variations in the grid conditions (e.g. in the number of errors) can have a significant impact on the makespan. To have more insight about our experiments, we also measured data transfer times and the number of errors as reported on Table 2. Moreover, in Figures 8-(d,e,f), 10-(d,e,f) and 11-(d,e,f) we also plotted the makespan[5] as a function of the average number of registered pilots during the simulation. These figures show that, even if the makespan is influenced by the number of registered pilots (dependent upon grid conditions), overall the difference in performance can indeed be attributed to the different implementations and splitting methods.

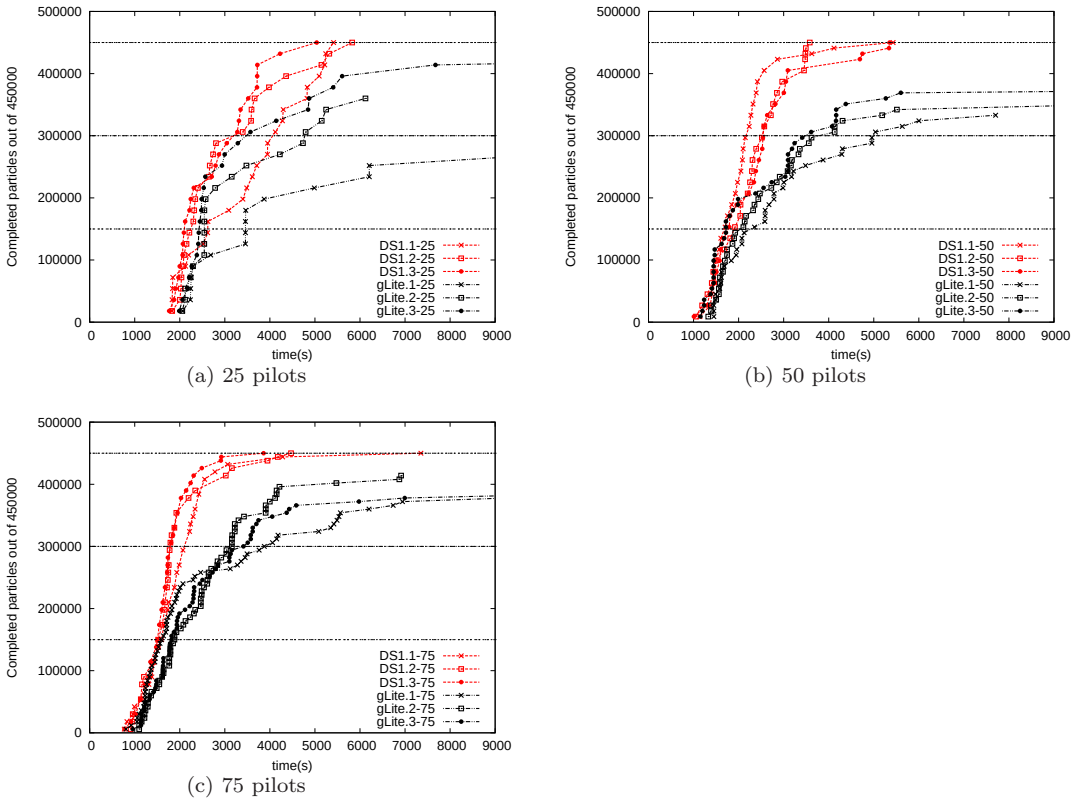
Although special care was taken to ensure that experiments are run in similar conditions, grid conditions on a production system like EGEE cannot be entirely controlled. For a reliable interpretation of our results, each experiment was repeated 3 times and measurements were done on multiple factors that could have influenced the results. Nevertheless further benchmarking needs to be envisaged for a finer (quantitative) analysis of the presented approaches. In the following we compare the scenarios two by two by analyzing their performance and reliability.

4.2.1 Pilots impact

In order to evaluate the gain of pilot jobs w.r.t. regular gLite WMS submission, we compare scenario 1 (gLite) with scenario 2 (DS1). Figure 7 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 EGEE resources. From 33% to 66%, scenario 1 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 exploiting 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/pilots). As expected, pilot jobs bring dramatic improvement w.r.t. default gLite, both in terms of reliability and performance.

For the DS scenario, the simulation completion rate begins to significantly slow down around 400,000 particles (90%). This is due to late resubmission of failed tasks or assignment of tasks to slow resources towards the end of the simulation. This corresponds to the issue of the last tasks as presented in Section 2 and will be addressed by the dynamic partitioning method as the results in the next section show.

[5] In Figure 8(d) due to an experimental problem (temporary connectivity issues with the master) we were unable to determine the makespan of DS2.1-25



(a) 25 pilots

(b) 50 pilots

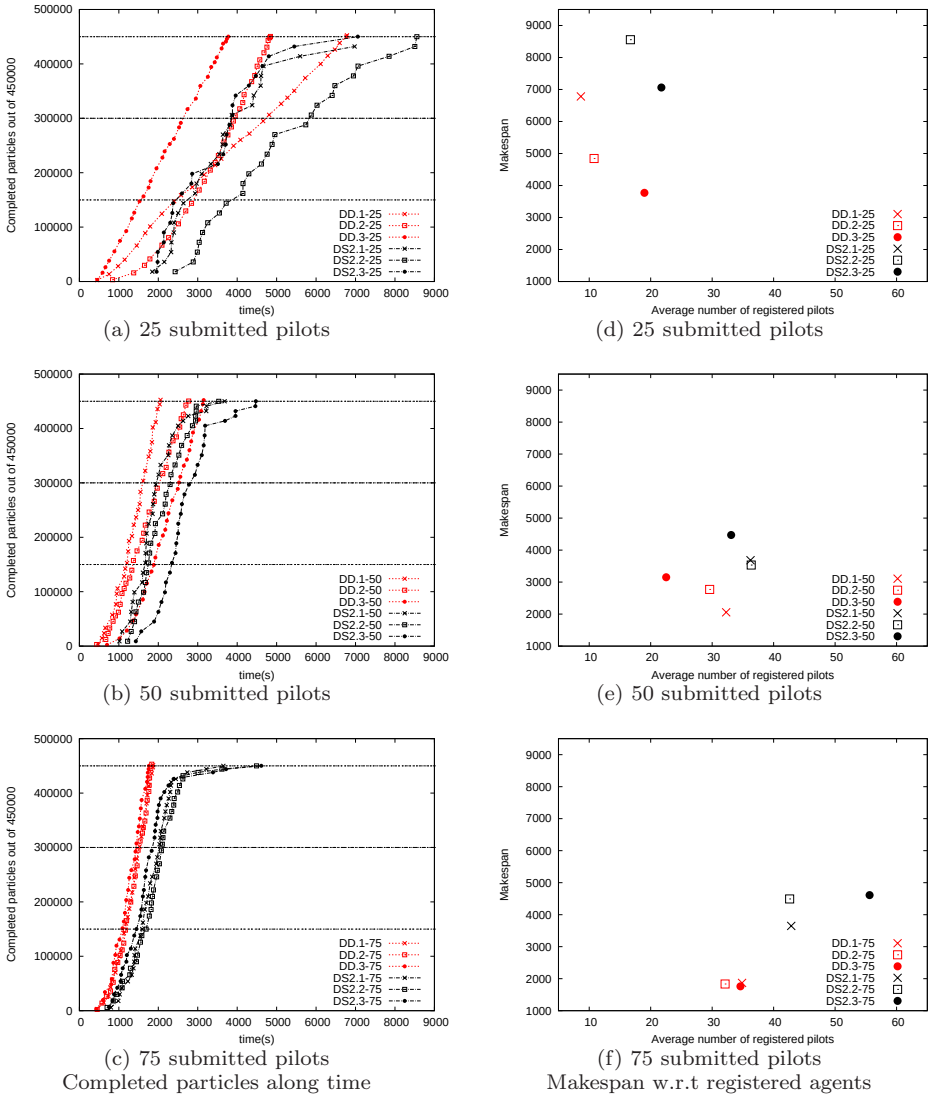
(c) 75 pilots

Fig. 7 DIANE-only implementation vs regular gLite-WMS implementation. gLite performance degrades drastically after 66% and never manages to reach 100% of the simulation.

4.2.2 Dynamic parallelization impact

A comparison between the dynamic and static partitioning approaches can be seen in Figure 8 where DS2 and DD experiments are plotted. 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, no slowdown is observed towards the end of the simulation for the dynamic approach. In Figure 8 (a), the poor performance of the DD.1-25 experiment is due to the small number of registered pilots as noticed in Section 4.2.

In Figure 8-(d,e,f) we can observe the makespan variability with respect to the number of registered pilots. The latter mainly depends on the grid conditions, but also on the simulation makespan. Given the same simulation, but different grid conditions, the simulation will finish faster if the grid conditions are favorable, i.e. if the largest number of pilots are available as soon as possible. Given the same number of pilots available along time (and knowing that pilots register progressively as shown in Figure 6) and two different splitting approaches, the faster approach will finish earlier, and thus fewer pilots have the chance to register. In Figure 8 (d,e,f) we see clearly



(a) 25 submitted pilots (d) 25 submitted pilots

(b) 50 submitted pilots (e) 50 submitted pilots

(c) 75 submitted pilots (f) 75 submitted pilots

Completed particles along time Makespan w.r.t registered agents

Fig. 8 DIANE-only dynamic implementation vs DIANE-only static implementation. The dynamic parallelization brings significant performance improvement as the makespan is considerably smaller.

that despite the variations in the number of average registered pilots, the makespans for the dynamic partitioning approach are (much) lower than for the static approach. This proves that the difference in performance is mainly determined by the splitting method and not by the grid conditions. We also notice that for the same splitting method, performance is better (makespan is smaller) when more pilots register. This rule applies less for the static approach, where the simulation makespan highly depends

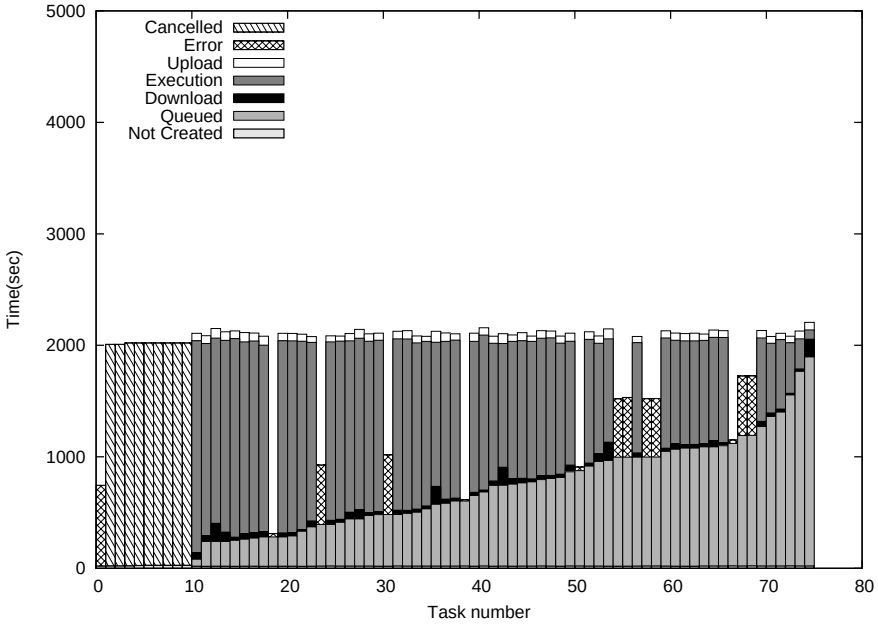


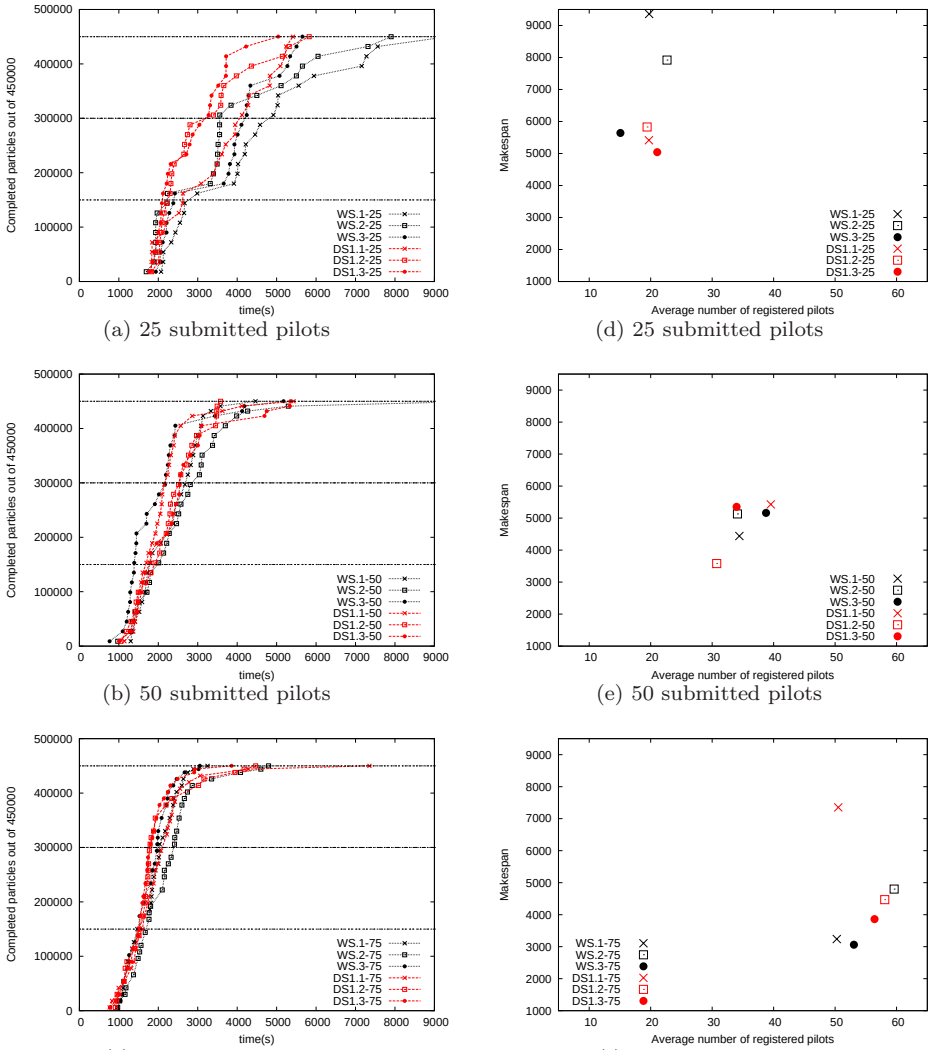
Fig. 9 Example of task flow obtained with dynamic partitioning of a GATE simulation on EGEE with 75 submitted pilots. Available resources are all exploited until the end of the simulation. Errors are compensated without any resubmissions. The scheduling obviously outperforms the one obtained with static partitioning (Figure 1).

on the slower tasks and failures, being thus very sensitive to grid conditions and less predictable.

Figure 9 plots a task flow obtained with dynamic partitioning. The scheduling obviously outperforms the one obtained with static partitioning (see Figure 1). Tasks complete almost simultaneously and errors are compensated without any resubmissions. The dynamic partitioning succeeds in compensating the problems (errors, spare or slow resources) of the static approach by exploiting available resources until the end of the simulation. The result is outstanding in terms of scheduling as tasks complete almost simultaneously.

4.2.3 Workflow overhead

Figure 10 compares the workflow versus the DIANE-only implementation for static task partitioning and Figure 11 compares the workflow versus the DIANE-only implementation for dynamic task partitioning. The two implementations have similar performance, with no significant difference for 50 and 75 pilots. For 25 pilots (longest tasks) DIANE-only has better makespan than the workflow implementation in the static case. In this case, time to 33% is almost identical but workflow is then slowed down by a higher number of errors as can be seen in Table 2. Reliability is very good in both cases, but workflow implementation has overall more errors than DIANE-only because of SE usage. Output transfer errors are very penalizing because they are detected late. These results are confirmed for all three cases (25, 50 and 75 jobs/pilots).



(a) 25 submitted pilots (d) 25 submitted pilots

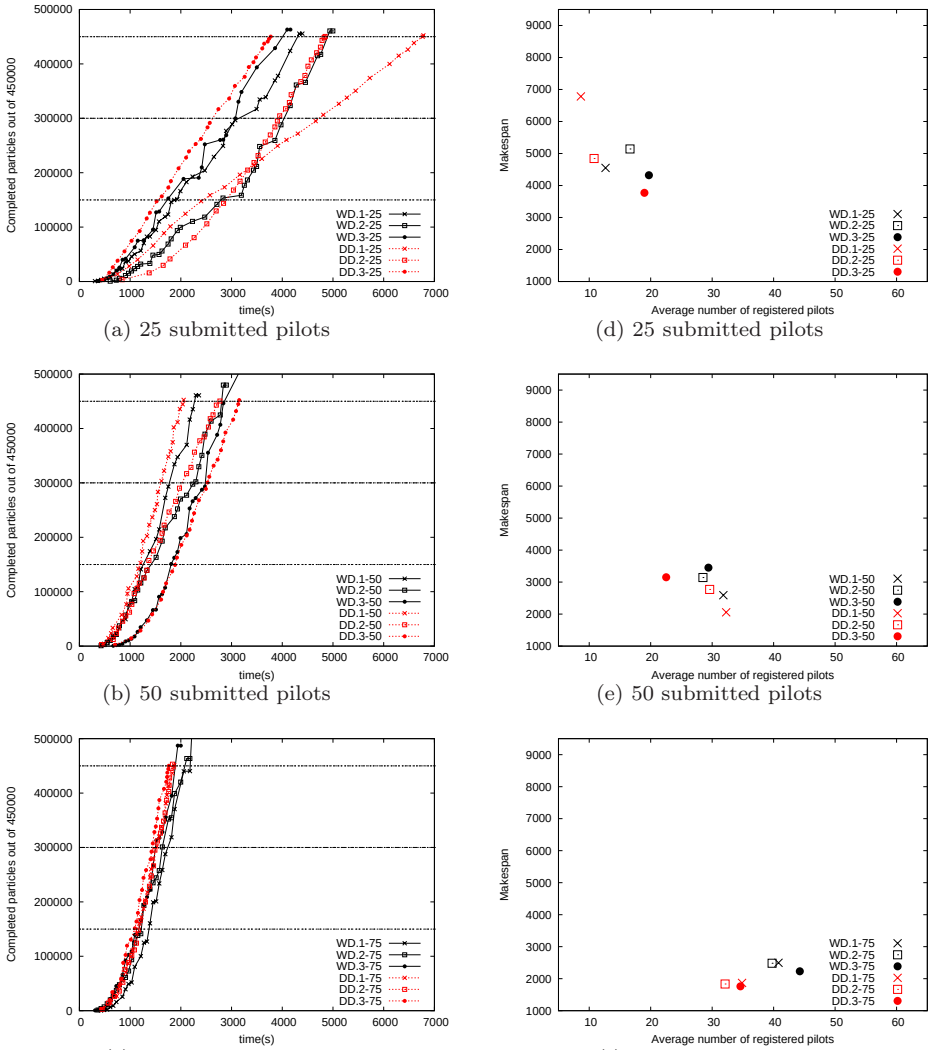
(b) 50 submitted pilots (e) 50 submitted pilots

(c) 75 submitted pilots (f) 75 submitted pilots

Completed particles along time Makespan w.r.t registered agents

Fig. 10 Workflow implementation vs DIANE-only static approach. Workflow implementation provides a generic framework with no significant performance loss.

Transfer time, also available in Table 2, is significantly smaller in the DIANE only approach as compared to the workflow approach because of the use of SEs. However, this difference does not seem to have a measurable impact on the overall performance. Overall, we conclude that the workflow implementation does not significantly penalize the execution.



(a) 25 submitted pilots (d) 25 submitted pilots

(b) 50 submitted pilots (e) 50 submitted pilots

(c) 50 submitted pilots (f) 50 submitted pilots

Completed particles along time Makespan w.r.t registered agents

Fig. 11 Workflow implementation vs DIANE-only dynamic approach. Workflow implementation provides a generic framework with no significant performance loss.

5 Conclusion

This paper presented dynamic partitioning and execution of GATE, an open-source software for nuclear medicine and radiation therapy simulations, on the EGEE grid. As a particle tracking Monte-Carlo system, GATE is naturally parallelizable and different parallelization methods can be found in the literature for its execution on distributed environments. However, these methods are not perfectly adapted for large-scale pro-

DIANE-only Scenario	Transfer time	Errors		Workflow Scenario	Transfer time	Errors
DS1.1-25	1156	5		WS.1-25	2488	7
DS1.2-25	1842	1		WS.2-25	2198	8
DS1.3-25	648	3		WS.3-25	2114	4
DS1.1-50	3064	2		WS.1-50	5395	11
DS1.2-50	1332	6		WS.2-50	6027	12
DS1.3-50	3492	6		WS.3-50	4650	9
DS1.1-75	3245	13		WS.1-75	6063	18
DS1.2-75	3551	11		WS.2-75	7229	24
DS1.3-75	5200	7		WS.3-75	8692	14
DD.1-25	3418	1		WD.1-25	1682	0
DD.2-25	2537	1		WD.2-25	1848	6
DD.3-25	2125	3		WD.3-25	2016	6
DD.1-50	1957	2		WD.1-50	4935	6
DD.2-50	2010	8		WD.2-50	4953	8
DD.3-50	1910	7		WD.3-50	5373	4
DD.1-75	2676	5		WD.1-75	7956	6
DD.2-75	2562	1		WD.2-75	4872	9
DD.3-75	2098	2		WD.3-75	6183	13

Table 2 Total transfer times (upload+download) and errors for DIANE-only implementation (left-hand side columns) vs Workflow implementation (right-hand side columns)

duction infrastructures like EGEE. Moreover, the heterogeneity of EGEE leads to recurrent errors and high latencies which lead to reduced performance.

To address these problems, we proposed a new dynamic splitting method for GATE and two implementations for its integration on EGEE. Results show that the proposed algorithm brings significant improvement w.r.t conventional static splitting. Indeed, by using our dynamic method, simulations complete up to two times faster than with the static partitioning. The dynamic approach achieves significantly better scheduling, all tasks completing almost simultaneously. Also, the two proposed implementations largely outperform the classical gLite job submission, by achieving 100% of the results without job/pilot resubmission and by significantly lowering the completion time. The workflow implementation provides a generic framework for the integration of new applications with different computing models and their execution on other systems. Results show that this generic framework is not penalizing in terms of performance.

The work presented is already in production for the (non-clinical) radiation therapy researchers at Creatis. As future work we plan to make it available for other research teams worldwide. Moreover, we want to extend the algorithm and implementations proposed here to other Monte-Carlo applications. We also envisage investigating the possibility of a multi-platform parallelization on clusters and GPUs and dynamically distribute the charge among these heterogeneous platforms.

Acknowledgements This work is co-funded by the European Commission through the EGEE-III project [6], contract number INFSO-RI-222667, and the results produced made use of the EGEE grid infrastructure. This work is also in the scope of the scientific topics of the French National Grid Institute (IdG). The authors would like to thank the EGEE site administrators and the ggus support for their work, Fabrice Bellet for his help with application compiling and customization for the grid, as well as the reviewers for their sharp and useful reviews which helped improving the paper.

[6] www.eu-egee.org

References

1. Ahn, S., Namgyu, K., Seehoon, L., Soonwook, H., Dukyun, N., Koblitz, B., Breton, V., Sangyong, H.: Improvement of Task Retrieval Performance Using AMGA in a Large-Scale Virtual Screening. In: NCM'08, pp. 456–463 (2008)
2. Allison, J., Amako, K., Apostolakis, J., Araujo, H., Arce 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., Mendez 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., Yoshida, H.: Geant4 developments and applications. Nuclear Science, IEEE Transactions on **53**(1), 270–278 (2006)
3. Bagnasco, S., Betev, L., Buncic, P., Carminati, F., Cirstoiu, C., Grigoras, C., Hayrapetyan, A., Harutyunyan, A., Peters, A.J., Saiz, P.: Alien: Alice environment on the grid. Journal of Physics: Conference Series **119**(6) (2008)
4. Cirne, W., Brasileiro, F., Paranhos, D., Goes, L., Voorsluys, W.: On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. Parallel Computing **33**, 213–234 (2007)
5. 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., Katz, D.S.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal **13**(3), 219–237 (2005)
6. Galyuk, Y.P., Memnonov, V., Zhuravleva, S.E., Zolotarev, V.I.: 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, pp. 515–520. Springer-Verlag, London, UK (2002)
7. Glatard, T., Montagnat, J., Lingrand, D., Pennec, X.: 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 (2008)
8. 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., Breton, V.: Grid-enabled Virtual Screening against malaria. Journal of Grid Computing (JGC) **6**(1), 29–43 (2008)
9. 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., Breton, V.: Grid enabled virtual screening against malaria. Journal of Grid Computing **6**, 29–43 (2008)
10. Jan, S., Santin, G., Strul, D., Staelens, S., Assi, K., Autret, D., Avner, S., Barbier, R., Bardis, M., Bloomfield, P.M., Brasse, D., Breton, V., Bruyndonckx, P., Buvat, I., Chatziioannou, A.F., Choi, Y., Chung, Y.H., Comtat, C., Donnarieix, D., Ferrer, L., Glick, S.J., Groiselle, C.J., Guez, D., Honore, P.F., Kerhoas-Cavata, S., Kirov, A.S., Kohli, V., Koole, M., Krieguer, M., van der Laan, D.J., Lamare, F., Largeron, G., Lartizien, C., Lazaro, D., Maas, M.C., Maigne, L., Mayet, F., Melot, F., Merheb, C., Pennacchio, E., Perez, J., Pietrzyk, U., Rannou, F.R., Rey, M., Schaart, D.R., Schmidtlein, C.R., Simon, L., Song, T.Y., Vieira, J.M., Visvikis, D., de Walle, R.V., Wiers, E., Morel, C.: GATE: a simulation toolkit for PET and SPECT. Phys Med Biol **49**(19), 4543–4561 (2004)
11. Kacsuk, P., Farkas, Z., Fedak, G.: Towards making BOINC and EGEE interoperable. In: 4th eScience Conference, pp. 478–484. Indianapolis (2008)
12. Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. Journal of Grid Computing (JGC) **3**(3-4), 221 – 238 (2005)
13. 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., Wilson, A.: Programming the Grid with gLite. Computational Methods in Science and Technology **12**(1), 33–45 (2006)

14. Maeno, T.: Panda: distributed production and distributed analysis system for atlas. Journal of Physics: Conference Series **119**(6), 062,036 (4pp) (2008)
15. Maheshwari, K., Missier, P., Goble, C., Montagnat, J.: Medical Image Processing Workflow Support on the EGEE Grid with Taverna. In: Intl Symposium on Computer Based Medical Systems(CBMS'09). IEEE (2009)
16. Maigne, L., Hill, D., Calvat, P., Breton, V., Lazaro, D., Reuillon, R., Legré, Y., Donnarieix, D.: Parallelization of Monte-Carlo simulations and submission to a grid environment. In: Parallel Processing Letters HealthGRID 2004, vol. 14, pp. 177–196. Clermont-Ferrand France (2004)
17. Mascagni, M., Li, Y.: Computational infrastructure for parallel, distributed, and grid-based Monte-Carlo computations. In: LSSC, pp. 39–52 (2003)
18. Moscicki, J.T.: Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data. In: Nuclear Science Symposium Conference Record, 2003 IEEE, vol. 3, pp. 1617–1620 Vol.3 (2003)
19. Moscicki, J.T.: Distributed analysis environment for HEP and interdisciplinary applications. Nuclear Instruments and Methods in Physics Research A **502**, 426429 (2003)
20. Moscicki, J.T., Brochu, F., Ebke, J., Egede, U., Elmsheuser, J., Harrison, K., Jones, R., Lee, H., Liko, D., Maier, A., Muraru, A., Patrick, G., Pajchel, K., Reece, W., Samset, B., Slater, M., Soroko, A., Tan, C., van der Ster, D., Williams, M.: Ganga: A tool for computational-task management and easy access to grid resources. Computer Physics Communications (2009)
21. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics journal **17**(20), 3045–3054 (2004)
22. Olabarriaga, S., de Boer, P.T., Maheshwari, K., Belloum, A., Snel, J., Nederveen, A., Bouwhuis, M. (eds.): Virtual Lab for fMRI: Bridging the Usability Gap. IEEE, Amsterdam (2006)
23. Olabarriaga, S., Glatard, T., de Boer, P.T.: A virtual laboratory for medical image analysis. IEEE Transactions on Information Technology In Biomedicine (TITB) (2010)
24. Procassini, R., O'Brien, M., Taylor, J.: Load Balancing of Parallel Monte Carlo Transport Calculations. In: Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications. Palais des Papes, Avignon, Fra (2005)
25. Germain Renaud, C., Loomis, C., Moscicki, J., Texier, R.: Scheduling for Responsive Grids. Journal of Grid Computing **6**, 15–27 (2008)
26. Reuillon, R., Hill, D., Gouinaud, C., El Bitar, Z., Breton, V., Buvat, I.: Monte Carlo Simulation With The GATE Software Using Grid Computing. In: Proceedings of NOTERE 2008 8ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition, NOTERE 2008. Lyon France (2008)
27. Sfiligoi, I.: glideinWMS - A generic pilot-based workload management system. Journal of Physics: Conference Series **119**(6), 062,044 (9pp) (2008)
28. 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., Jimenez, S.G.: Dirac: a community grid solution. Journal of Physics: Conference Series **119**(6), 062,048 (12pp) (2008)