A Genetic Algorithm with Communication Costs to Schedule Workflows on a SOA-Grid

Jean-Marc Nicod, Laurent Philippe, and Lamiel Toch

LIFC, Laboratory, Université de Franche-Comté, Besançon, France

Abstract. In this paper we study the problem of scheduling a collection of workflows, identical or not, on a SOA (Service Oriented Architecture) grid . A workflow (job) is represented by a directed acyclic graph (DAG) with typed tasks. All of the grid hosts are able to process a set of typed tasks with unrelated processing costs and are able to transmit files through communication links for which the communication times are not negligible. The goal of our study is to minimize the maximum completion time (makespan) of the workflows. To solve this problem we propose a genetic approach. The contributions of this paper are both the design of a Genetic Algorithm taking the communication costs into account and its performance analysis.

1 Introduction

Nowadays, to go further in their research, scientists often need to connect several applications together. Therefore, since few years, workflow systems have been designed to provide tools that support these multi-application simulations. In e-Science [16] many fields as medical image processing, geosciences or astronomy use workflow applications. A workflow is defined as a set of applications that are connected to each other by precedence constraints. An input data set enters the workflow, it is processed by an application which computes an output data set that is, in turn, sent to the next application of the workflow structure. Generally a workflow has the structure of a DAG (Direct Acyclic Graph): a graph whose nodes are tasks and whose edges are precedence constraints.

When the size of the data entering the workflow increases the processing time may become very long and it becomes necessary to use larger computing resources as grids. Because of their heterogeneity these platforms are however difficult to efficiently use for non computer scientists mainly due to the complexity of scheduling the tasks on the hosts. Several research projects have already tackled this problem. The Pegasus framework [8] proposes a convenient way for scientists to compute their workflows onto heterogeneous platforms without learning distributed programming concepts. Other tools, like DIET [2] or NINF-G [15], provide a SOA-Grid (Service Oriented Architecture) that facilitates user accesses to remotely accessible computing applications and make the execution of workflows on a heterogeneous platform easier.

When the number of workflows to be executed in parallel is large, we must efficiently map them onto the heterogeneous resources. Minimizing the execution time (makespan) of a workflow or a set of workflows onto a heterogeneous platform is however an optimization problem that is known to be NP-Hard [13]. As a consequence, the problem considered here is also NP-Hard and we can just rely on heuristics to find a solution as good as possible. Classical scheduling algorithms often rely on list-based scheduling algorithms. They use heuristics such as Heterogeneous Earliest Finish Time (HEFT) [17] or Critical Path [11]. In a homogeneous context, [12] gives a survey on the DAG scheduling topic and the study presented in [9] evaluates eleven heuristics, such as Min-min, Max-min or Sufferage. Makespan oriented strategies to schedule workflows onto the grid are presented in [14], with the description of real life medical applications. In [18], the problem of scheduling a set of different DAGs is studied. These approaches compute an off-line schedule considering the whole set of tasks. But, when the number of tasks scales up, the computation time becomes too long because of the complexity of the algorithm. Genetic Algorithms (GA) are known to give good results in several optimization domains and algorithms that schedules tasks are for instance presented in [10] and [5]. They are designed for scheduling tasks of one workflow onto a heterogeneous platform but without taking communication costs into account. Other studies tackle workflow scheduling onto heterogeneous platforms but with other objective functions than the makespan. The Steady-State technique presented in [1] provides an optimal algorithm to schedule a set of identical workflows also for the throughput objective function.

In this study, the general problem we deal with is to schedule a set of workflows (jobs) onto a SOA-grid. Each task of a workflow has a type that corresponds to a service type in the SOA-Grid. We consider the two following cases: (1) the general case where the structure of each workflow differs from one another; (2) the particular case when the structure of the workflow is the same for all jobs. In this paper we focus on researches that we carried out on the design of a Genetic Algorithm and we assess its performance to schedule a set of workflows. The contributions of this paper are both the design of a Genetic Algorithm taking the communication costs into account and the performance analysis. In the general case (1) we compare the performance obtained by a GA approach to the performance obtained by a list-based scheduling algorithm. In the particular case (2) of the scheduling of a collection of identical workflows which structure is limited to intrees, we compare our results to a lower bound and we show that our Genetic Algorithm approach allows us to get a schedule with good performance.

The paper is organized as follows. In Section 2 we detail the framework which defines both the grid and the workflow models. We present in Section 3 a Genetic Algorithm that takes communication costs into account to deal with our problem. Section 5 introduces the simulation setup, the results of the simulations and their analysis for workflows of different shape. Section 6 presents the simulations for workflows with identical structure. Finally, we conclude in Section 7.

2 Framework

In this section we formally define the context of our study.

Applicative Framework. Our problem that is to schedule a collection $\mathcal{B} = \{\mathcal{J}^j, 1 \leq j \leq N\}$ of N workflows. Each workflow $\mathcal{J}^j$ is represented by a DAG $\mathcal{J}^j = (\mathcal{T}^j, \mathcal{D}^j)$ where $\mathcal{T}^j = \{T_1^j, \ldots, T_{n_j}^j\}$ is the set of its tasks and $\mathcal{D}^j$ represents the precedence constraints between the tasks. These precedence constraints are files: $F_{k,i}^j$ is the file sent between T_k^j and T_i^j when $(T_k^j, T_i^j) \in \mathcal{D}^j$. Let $\mathcal{T} = \cup_{j=1}^N \mathcal{T}^j = \{T_{i_j}^j, 1 \leq i_j \leq n_j \text{ and } 1 \leq j \leq N\}$ be the set of tasks to schedule. We define $t(i,j)$ as the type of task T_i^j.

Target Platform. The target platform PF is a heterogeneous platform of n machines modeled by an undirected graph $\mathcal{PF} = (\mathcal{P}, \mathcal{L})$ where the vertices in $\mathcal{P} = \{p_1, \ldots, p_n\}$ represent the machines and where the edges of $\mathcal{L}$ are the communication links between machines.

Let τ be the set of all task types available onto the platform. Each machine p_i is able to perform a subset of τ. If the type $t \in \tau$ is available on the machine p_i, $w(t, p_i)$ is the time to perform a task of type t on p_i. Moreover, each link (p_i, p_j) has a bandwidth $bw(p_i, p_j)$ which is the number of data per time unit that can be transferred through that link. We define $a(i,j)$ such that $p_{a(i,j)}$ is the machine on which T_i^j is assigned.

Communication Model. In our study the processors are interconnected by communication links in a point-to-point fashion to model a computation grid. In the literature [1,3], several communication models exist such as the one-port model or the multi-port model. We choose to use the one-port model because of its ease of modeling and of implementation while still being realistic. In this model only one data can be transmitted at the same time over a communication link and a node can do at most one reception and one transmission at the same time. We define $\mathcal{R}(p_k, p_i) = \{(p_j, p_{j'}) \in \mathcal{L}\}$ as a route from p_k to p_i.

3 GA with Communication Costs

For the execution we use the same genome representation as in [5,10], i.e., a chromosome is a two-dimension table with one row per node where the tasks assigned to the node are recorded. Some improvements to take SOA-Grids and collections into account, presented in [7], are added. As these GA coding does not however take communication costs into account, we study their integration in this section.

Communication Integration. The main issue we faced is the step of the genetic algorithm where we should integrate the communications.

The first solution we have studied is to add communications in chromosomes, as for executions. We thus introduce the notion of "*communication task*" to represent file exchanges in the task graph and we map them onto communication links. The chromosome thus becomes a two-dimension table with one row per node and one row per communication link. Tasks are recorded in the node's rows and communication are recorded in the link's rows. This leads however to issues regarding both the crossover and the mutation genetic procedures. As the choice of a communication link depends on the source and sink nodes not every link can be used for a given communication. So these procedures generate inconsistent communication routes and so numerous non feasible schedules that waste a lot of computation time.

A more convenient way to introduce communications is in the fitness evaluation step of the genetic algorithm. Indeed the allocations of the communication tasks depend on the allocations of the computation tasks since the possible routes between two nodes are limited. Choosing a good mapping for the computation tasks intuitively involves finding a valid allocation for the communication tasks. So we use the chromosomes only to map the computation tasks onto the nodes, and in a second step to look for a valid route to send the data according to this mapping. The fitness of each individual is then evaluated by algorithm 1. For each task of the chromosome it computes its start time depending on the processor avalaibility (lines 5-7) and when the used processor will become iddle (lines 8-9). The algorithm then remove the task for the set of remaining tasks (line 10). At the end it returns the fitness of the chromosome.

Algorithm 1. Computing the fitness of a chromosome

```
   Input  : $PF$: the platform and $B$: collection of workflows
   Output: $f(ch)$: the fitness of the task with chromosome $ch$
   Data: $\mathcal{T}_{ToSched}$: the set of remaining tasks to schedule, $C(T_i^j)$: the completion
         time of $T_i^j$, $\sigma(T_i^j)$: the start time of $T_i^j$ on $p_{a(i,j)}$, $\delta(p_u)$: the next time $p_u$
         is idle, $p_{a(i,j)}$ the machine on which $T_i^j$ is assigned, $w(t, p_i)$: the time to
         perform a task of type $t$ on $p_i$, $CT(F_{k,i}^j)$: the communication time to
         send $F_{k,i}^j$ along route $\mathcal{R}(p_{a(k,j)}, p_{a(i,j)})$
1  $\mathcal{T}_{ToSched} \leftarrow \mathcal{T}$
2  while $\mathcal{T}_{ToSched} \neq \emptyset$ do
3      choose a free task $T_i^j \in \mathcal{T}_{ToSched}$ (Earliest Finish Time heuristic)
       $\mathcal{T}_{pred} \leftarrow \{T_k^j | (T_k^j, T_i^j) \in \mathcal{D}^j\}$
4      $\sigma(T_i^j) \leftarrow 0$
5      foreach task $T_k^j \in \mathcal{T}_{pred}$ do
6          $\sigma(T_i^j) \leftarrow \max(\sigma(T_i^j), C(T_k^j) + CT(F_{k,i}^j))$
7      $\sigma(T_i^j) \leftarrow \max(\delta(p_{a(i,j)}), \sigma(T_i^j))$
8      $C(T_i^j) \leftarrow \sigma(T_i^j) + w(t(i,j), p_{a(i,j)})$
9      $\delta(p_{a(i,j)}) \leftarrow C(T_i^j)$
10     $\mathcal{T}_{ToSched} \leftarrow \mathcal{T}_{ToSched} \setminus \{T_i^j\}$
11 return $f(ch) = 1/C_{max} = 1/max_{T_i^j \in \mathcal{T}}(C(T_i^k))$
```

4 Simulation Setup

To assess the performance of the scheduling algorithm we need to implement it on a heterogeneous platform. The context is indeed too complex to be studied with a formal approach and, to get realistic results, the platforms used must integrate the network contention. On the other hand, the implementation on a computation grid can however not give reproducible results as the experimental conditions, as the network load, may change. So, we use a grid simulator to evaluate the performance of the Genetic Algorithm. The simulator is implemented using SimGrid and its MSG API [4].

All simulations have been made using batch sizes from 1 to 10 000. The platforms and the applications are randomly generated with a uniform distribution. Platforms have between 4 and 10 nodes and are strongly connected. Applications have between 4 and 12 tasks. For the case presented in Section 5, where workflows are different from each other, 200 platforms and 10 000 DAGs are randomly generated. For each couple (platform, batch size) different applications are randomly chosen among the 10 000 ones. So we generate 1 900 simulations of platform/application scenarios. In the case where workflows are identical, we use 10 platforms and 10 applications, so 100 scenarios for each batch size.

We define the "*communication to computation ratio*" (CCR) of a simulation as the average computation time divided by the average communication time. To assess the impact of the communications on the algorithm performance, we run simulations with different CCR, from 1 000 (i.e., communication time is predominant) to 1/1 000 (i.e., communication time is negligible). The speed of the nodes are unrelated. We also assess the impact of the platform heterogeneity on the performance: execution and communication times fluctuate respectively in a range from 1 to 10 and from 1 to 4.

For the GA, the population is set to 200 individuals and a generation is set to 100 iterations.

5 Results with General DAGs

As the execution time of a collection of workflows cannot be computed in polynomial time, we must compare the performance of the GA to another algorithm: we use a standard list-based scheduling algorithm. The results presented in this section concern fully heterogeneous platforms and $CCR \approx 1$.

Impact of the Communication Model. Choosing the network links for each communication is complex. Always using the shortest (or the fastest) path may lead to a high contention so that we need to take the network load into account. We assess here four communication policies:

– "GA-no-comm": the route between two nodes is statically and arbitrary chosen and the communication costs are not used to compute the fitness.
– "GA static-route": the same as the previous policy but the communication costs are used in the fitness computation.
– "GA 1-route-Bellman-Ford": the static route for each couple of computation nodes is the fastest one by using the Bellman-Ford algorithm.
– "GA 3-routes-Bellman-Ford": for each couple of computation nodes, during the evaluation fitness step, the 3 best routes are tested.

Figures 1a and 1b show the efficiencies of the GA with different communication integrations. The notion of relative measure to optimal (RMO) is introduced in Section 6, however, we just need to know that the higher the curve, the more

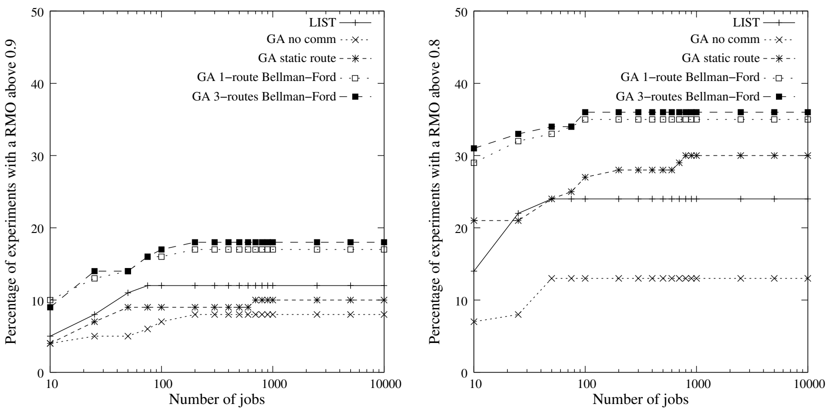
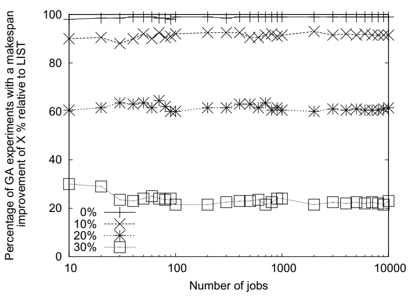


Fig. 1. Communication integrations with a RMO above 0.9 and 0.8

efficient is the algorithm. Comparing these policies to a list-based scheduling (LIST) allows us to remark on one hand that this is mandatory to take communication into account to increase the performance but on the other hand that the use of dynamic information in *GA 3-routes-Bellman-Ford* does not significantly improve the performance. We nevertheless use this implementation, as we get the best results for the performance analysis in the following.



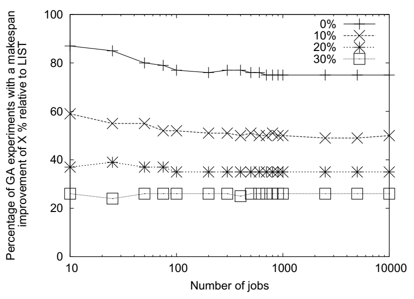


Fig. 2. Improvement for different DAGs **Fig. 3.** Improvement for identical DAGs

Figure 2 shows the improvement of the makespan for a set of different workflows on a heterogeneous platform. Each curve shows the percentage of GA experiences whose makespan is improved respectively by more than 0%, 10%, 20% and 30% when compared to the list-based scheduling algorithm (LIST). We notice that more than 80% of the GA experiments have a makespan improved by more than 10%. The 0%-curve shows that GA never slows down the performance.

Figure 3 shows the improvement of the makespan for a set of identical workflows. We can notice that GA gives less important improvements relative to LIST in this case. For each 0, 10, 20 and 30%-curve, GA improvements decrease by about 10% compared to experiments obtained when scheduling different workflows. We can also note that 10% to 20% of the GA schedules are less efficient than the LIST schedules in this case. GA improvements remain however high.

In general, more than 70% of the GA experiments give a makespan with an improvement greater than 10% for almost any size of sets of workflows. The diversity of the workflows promotes the GA algorithm: as GA tries randomly a lot of combinations, it can benefit from different workflows while LIST which is directed by greedy choices, cannot.

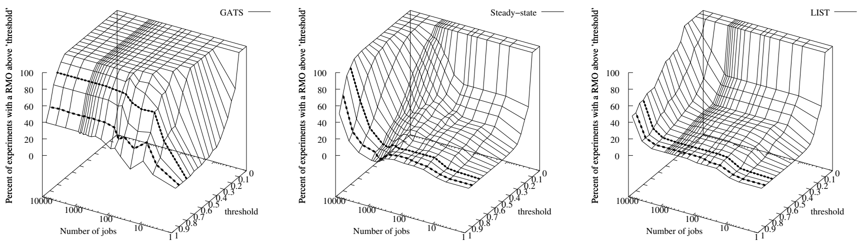
6 Results for a Set of Identical Intrees

Comparing two algorithms gives relative information but no absolute information on the quality of the algorithm. For the particular case of batches of intrees it is possible to compute an optimal throughput by using a Steady-State algorithm [1]. In this section we assess how far the GA is from the optimal and we compare it to the list scheduling and to a practical makespan oriented implementation of the Steady-State algorithm [6]. The latter reaches the optimal solution for an infinite number of jobs, as its objective function is the throughput, but limits this study to the particular case of intree-shaped workflows.

Relative Measure to Optimal. The optimal throughput ρ can be computed thanks to the Steady-State algorithm using linear programming. This optimal throughput is used to compute a lower bound L_0 for the optimal makespan as the number of jobs to be processed over the throughput ($L_0 = \frac{N}{\rho}$). Let $makespan_o$ be the optimal makespan, then $makespan_o \geq L_0$. To evaluate the algorithm performance, we introduce the notion of Relative Measure to Optimal (RMO) as the ratio between this lower bound L_o over the makespan $makespan_r$ of the schedule given by the algorithm ($RMO = \frac{\mathrm{L}_o}{makespan_r}$). So, the nearer RMO is from 1.0, the more efficient is the scheduling algorithm.

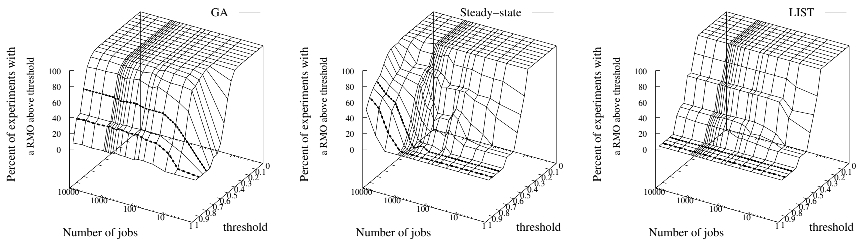
Since we run 100 simulations, for each of the 19 sizes of job sets, we cannot get a simple scalar RMO value for the overall experiments. A mean value would indeed be not meaningful as the longer simulations will weight more than the sorter ones with this metric. So we compute the percentage of experiments that gives a RMO greater than t for each size of batches. In the following we present the distribution of the results depending on this percentage on the 3D curves. Two lines are thickened on the surfaces to highlight the RMO curves for threshold values 0.8 and 0.9. They represent the quality of the schedule.

Fully Homogeneous Platforms. Figures 4a, 4b and 4c give the results for homogeneous platforms (homogeneous nodes and homogeneous communication links) and computation intensive applications ($CCR \approx 0.01$). In that case the GA algorithm give almost optimal results for 85% of the experiments. Figures 5a, 5b and 5c show the results for homogeneous platforms and communication intensive applications ($CCR \approx 1$). In that case the GA algorithm gives the best results for batches with less than 2 500 jobs and is overtaken by the Steady-State algorithm for jobs with more that 10 000 jobs. It also reaches the optimality for



(a) GA algorithm (b) Steady-State algorithm (c) LIST algorithm

Fig. 4. Simulation with fully homogeneous platforms, $CCR \approx 0.01$



(a) GA algorithm (b) Steady-State algorithm (c) LIST algorithm

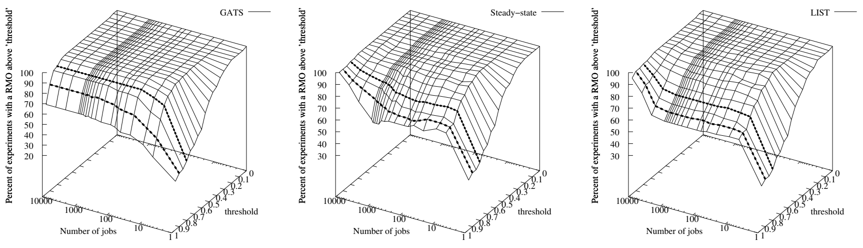
Fig. 5. Simulation with fully homogeneous platforms, $CCR \approx 1$

almost 70% of the experiments. From these experiments we can conclude that the GA algorithm performs globally well, much better than the simple LIST algorithm. The results are however lowered when we introduce communications because it makes the problem more complex.

Fully Heterogeneous Platforms. Figures 6a, 6b and 6c show the results obtained for a heterogeneous platform with $CCR \approx 0.01$ and Figures 7a and 7c show the results with $CCR \approx 1$. In this communication intensive case, no algorithm gives good results due to the complexity of the problem. The Steady-State tends slowly toward optimality. The GA nevertheless gives the best results for batches with less than 2 000 jobs. In the computation intensive case (i.e., $CCR \approx 0.01$), the GA performs the best compared to the other algorithms and its performance is not far from the homogeneous case.

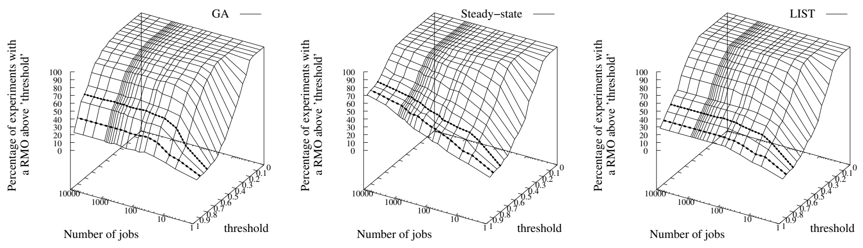
On the whole set of curves, we can note that the GA globally gives good schedules and it out performs the other algorithms for the studied context. It usually reaches its best performance from a few tens of jobs and stays stable for more jobs. So, in all of the studied cases, GA is a good choice for batches from one to a few thousand jobs. This latter limit depends on the properties of the platform.

Computation Times. The simulations have been run on a 2.8 GHz Intel Xeon bi-processor. The time needed to compute a schedule using the GA varies linearly according to the batch size: about 1 second per DAG. The CPU time needed



(a) GA algorithm (b) Steady-State algorithm (c) LIST algorithm

Fig. 6. $CCR \approx 0.01$, fully heterogeneous platforms



(a) GA algorithm (b) Steady-State algorithm (c) LIST algorithm

Fig. 7. Simulation with fully heterogeneous platforms, $CCR \approx 1$

by LIST varies linearly as well. It is about 5 times faster than GA. Finally, for Steady-State, the CPU time is very low for any batch size : about 75 seconds for scheduling 1 000 DAGs. So the GA and LIST are very time consuming while the computation time spent by the Steady-State algorithm is very low. Nevertheless, for large sizes of batches, the GA computation takes usually less than 20 minutes.

7 Conclusion and Future Work

In this paper, we propose a genetic algorithm that solves the problem of scheduling a collection of workflows on a set of heterogeneous nodes interconnected by heterogeneous communication links. This GA takes the communication costs into account. We show that for a collection of different workflows, GA obtains better execution performance than a classical LIST algorithm. For the case where the DAGs are identical intrees we are able to compare the GA results to an optimal lower bound and to show that its results tend towards the optimality for more than 1 000 jobs. The obtained results are moreover comparable to a practical implementation of the Steady-State algorithm. The main idea that we keep in mind is that scheduling with GA by taking communications into account is difficult, as for implementing the practical Steady-State solution. For future work, other communication models should be implemented in the simulator, like the multi-port models, and other genetic representations could be explored.

The huge amount of simulations have been run on the cluster of the *Mésocentre de Calcul de Franche-Comté* in Besançon, France.

References

1. Beaumont, O., Legrand, A., Marchal, L., Robert, Y.: Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms. In: HeteroPar 2004, pp. 296–302 (2004)
2. Caron, E., Desprez, F.: Diet: A scalable toolbox to build network enabled servers on the grid. IJHPCA 20(3), 335–352 (2006)
3. Casanova, H.: Modeling large-scale platforms for the analysis and the simulation of scheduling strategies. In: APDCM 2004 (2004)
4. Casanova, H., Legrand, A., Quinson, M.: Simgrid: A generic framework for large-scale distributed experiments. In: UKSIM 2008, pp. 126–131 (2008)
5. Daoud, M., Kharma, N.: GATS 1.0: A Novel GA-based Scheduling Algorithm for Task Scheduling on Heterogeneous Processor Nets. In: Genetic And Evolutionary Computation Conference (2005)
6. Diakité, S., Marchal, L., Nicod, J.-M., Philippe, L.: Steady-State for Batches of Identical Task Trees. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 203–215. Springer, Heidelberg (2009)
7. Diakité, S., Nicod, J.-M., Philippe, L.: Comparison of batch scheduling for identical multi-tasks jobs on heterogeneous platforms. In: PDP 2008, Toulouse, France, pp. 374–378 (2008)
8. Deelman, E., et al.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming Journal 13, 219–237 (2005)
9. Braun, T.-D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. JPDC 61, 810–837 (2001)
10. Goh, C.K., Teoh, E.J., Tan, K.C.: A hybrid evolutionary approach for heterogeneous multiprocessor scheduling. Soft Comput. 13, 833–846 (2009)
11. Kwok, Y., Ahmad, I.: Dynamic critical-path scheduling: An effective technique for allocating task graphs to multi-processors. In: PDS, pp. 506 – 521 (1996)
12. Kwok, Y.-K., Ahmad, I.: Static Scheduling Algorithms for Allocating Task Graphs to Multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
13. Lenstra, J.K., Rinnooy Kan, A.H.G.: Complexity of scheduling under precedence constraints. Operations Research 26(1), 22–35 (1978)
14. Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling strategies for mapping application workflows onto the grid. In: HPDC 2005, NC, Triangle Park, USA, pp. 125–134 (July 2005)
15. Tanaka, Y., Takemiya, H., Nakada, H., Sekiguchi, S.: Design, implementation and performance evaluation of gridrpc programming middleware for a large-scale computational grid. In: GRID 2004, pp. 298–305 (2004)
16. Taylor, I.-J., Deelman, E., Gannon, D.-B., Shields, M.: Workflows for e-Science (2007)
17. Topcuouglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. In: PDS, pp. 260–274 (2002)
18. Zhao, H., Sakellariou, R.: Scheduling multiple DAGs onto heterogeneous systems. In: HCW 2006, Rhodes, Greece (2006)