A Hybrid Algorithm for DAG Application Scheduling on Computational Grids

Lyes Bouali[1], Karima Oukfif[2,3], Samia Bouzefrane[4(✉)], and Fatima Oulebsir-Boumghar[3]

[1] LARI Laboratory, UMMTO, Tizi Ouzou, Algeria
bouali.lyes@gmail.com
[2] Compute Science Department, UMMTO, Tizi Ouzou, Algeria
karima.oukfif@gmail.com
[3] LRPE Laboratory, USTHB, Bab Ezzouar, Algeria
fboumghar@usthb.dz
[4] CEDRIC Laboratory, CNAM, Paris, France
samia.bouzefrane@cnam.fr

Abstract. In the late three decades, grid computing has emerged as a new field providing a high computing performance to solve larger scale computational demands. Because *Directed Acyclic Graph* (DAG) application scheduling in a distributed environment is a NP-Complete problem, meta-heuristics are introduced to solve this issue. In this paper, we propose to hybridize two well-known heuristics. The first one is the *Heterogeneous Earliest Finish Time* (HEFT) heuristic which determines a static scheduling for a DAG in a heterogeneous environment. The second one is *Particle Swarm Optimization* (PSO) which is a stochastic meta-heuristic used to solve optimization problems. This hybridization aims to minimize the makespan (i.e., overall completion time) of all the tasks within the DAG. The experimental results that have been conducted under hybridization show that this approach improves the scheduling in terms of completion time compared to existing algorithms such as HEFT.

Keywords: Grid computing · Task scheduling · Directed acyclic graph · Heterogeneous earliest finish time algorithm · Particle swarm optimization algorithm · Makespan

1 Introduction

The deployment of high-speed networks and powerful computers has involved to new computing paradigms. Hence, while current hardware infrastructures are distributed in nature such as in grid computing, the underlying applications are composed of tasks distributed on different nodes. In fact, a grid computing is a set of geographically remote resources deployed across multiple nodes allowing their computational power and storage space to be shared. Grid resources are heterogeneous, dynamic and owned by various administrative organizations under locally defined policies. Grids are used in a variety of scientific applications such as in astronomy, geophysics and bioinformatics where a single and powerful parallel super computer [1] cannot resolve the large-scale application issues.

S. Boumerdassi et al. (Eds.): MSPN 2015, LNCS 9395, pp. 63–77, 2015.
DOI: 10.1007/978-3-319-25744-0_6

To take advantage of the potentials of grid computing, efficient scheduling algorithms are fundamentally important [2]. The task-scheduling problem refers to the mapping of the application tasks to a set of distributed resources or nodes. Because this problem is NP-Complete, various algorithms are proposed in the literature with different criteria in order to schedule efficiently application tasks.

Our contribution in this paper is twofold: firstly, we propose a scheduling approach based on the hybridization of two scheduling algorithms like HEFT and an adapted DPSO, called DPSO*, for the dependent-tasks scheduling problem. Secondly, we undertake some measurements that show that the hybridization approach improves the performances in terms of makespan. Makespan is the time difference between the start and finish of a sequence of tasks.

The remainder of this paper is organized as follows. Related works are presented in Sect. 2. Section 3 formalizes the scheduling problem. Section 4 describes the HEFT, our adapted DPSO* algorithms and their hybridization. Then, performance tests are discussed in Sect. 5. Finally, Sect. 6 concludes the paper with some perspectives.

2 Related Works

In [3], Casavant and Kuhl have proposed a taxonomy of scheduling algorithms for general-purpose parallel and distributed computing systems. Since grid computing has specific features, scheduling algorithms for grid computing fall into a subset of this taxonomy [2]. In fact, in [2], the authors classified grid scheduling algorithms depending on whether the grid scheduling algorithm is static or dynamic, distributed or centralized, cooperative or non-cooperative.

Due to the NP-Complete property of the scheduling algorithms and the difficulty to prove the optimality of a given solution, researchers tried to find sub-optimal solutions through heuristic approaches. When the relationship between the tasks within the grid application is considered, scheduling algorithms are dichotomized into independent and dependent task scheduling. Hence, in [4], a comparison between eleven heuristics used to schedule independent tasks is discussed. Among these heuristics, we can find *Opportunistic Load Balancing*, *Minimum Execution Time*, *Minimum Completion Time,* etc. Each of them aims to assign a task to a resource with an optimal completion time.

In the case of dependent task scheduling, also called workflow scheduling, a task precedence graph called *Directed Acyclic Graph* (DAG) is usually used to model the application scheduling. The nodes of the DAG represent the tasks and the directed edges represent the execution dependencies and the data communication between tasks [5]. There are two major types of scheduling, best-effort based and QoS constraint based scheduling. Supporting QoS scheduling algorithms are based on either deadline (time) or budget (cost) constraints and are at a very preliminary stage [6]. Best-effort based scheduling attempts to minimize the makespan using different approaches. These approaches can be classified into different heuristics such as list-scheduling, clustering, duplication-based algorithms, and meta-heuristics (guided random search methods) approaches.

List-scheduling heuristics are based on two steps: in the prioritizing phase, tasks are ordered in a list by assigning a priority for each task, while in there resource selection phase each selected task is scheduled on the resource that minimizes a predefined cost

function [7]. Various research works have been proposed in the literature under this type of heuristics such as HEFT and CPOP [7], FCP [8], DCP [9], DLS [10], and xDCP [11].

While the clustering approaches (DSC [12], CASS II [13], EZ [9], CTHP [14]) assign a group of inter-communicated tasks to the same cluster hence to the same resource, the duplication based-scheduling approaches (DSH [15], CPFD [16], TDS [17], BTDH [18], THAN [19]) duplicate tasks to assign them to idle-time slots within the resource, thus avoiding the data communication overhead.

Besides, meta-heuristics are stochastic algorithms dedicated to solve optimization problems. Using meta-heuristic approach, there is no guarantee to find a global optimum but it provides an approximation of this optimum in a reasonable time. Genetic Algorithms (GA) [20–24] are examples of meta-heuristics that are widely used, for the good solutions they provide. To overcome the high execution time taken by GA, Kennedy and Eberhart introduced the Particle Swarm Optimization (PSO) methodology in [25]. In the context of grid computing, PSO has been used by the authors of [26] to schedule independent tasks by transforming the continuous values of particles into discrete values thanks to the *Smallest Position Value* (SPV) rule. In [27], Liu et al. designed a fuzzy scheme based on discrete PSO to solve the independent job scheduling problem on computational grids. Izakian et al. [28] have proposed a version of discrete PSO for grid independent job scheduling.

In this paper, our aim is to build an optimal scheduling algorithm by adapting the PSO algorithm and then by hybridizing it with HEFT heuristic, in order to schedule DAG tasks in the context of grid computing. In fact, according to the PSO principle, since the particles are initialized randomly, our idea is to inject a particular particle initialized thanks to HEFT algorithm that is considered as a high-quality solution, in order to enhance PSO technique and hence to reduce significantly the convergence time. Before describing this hybridization, we formalize the scheduling problem in the next section.

3 Scheduling-Problem Formalization

A scheduling system is generally modeled thanks to an application, a platform and a scheduling-performance criterion. In our case, the criterion is the makespan. In the next sub-sections, we formalize each part of the scheduling model.

3.1 Application Formalization

Each application is modeled by a Direct Acyclic Graph $G = (V, E)$, where V is a set of v vertices representing tasks (jobs) T_i $(1 \leq i \leq v)$, and E a set of directed edges. An edge $(i, j) \in E$, corresponds to a dependence constraint between task T_i and T_j. T_i is an immediate parent task of T_j, and T_j the immediate child task of T_i. A child task cannot be executed until all of its parent tasks are completed. A task with no parent tasks is called an *entry task* and a task with no children tasks is called an *exit task*. We assume that only one entry and one exit tasks exist in the graph.

Data matrix, with $v \times v$ dimensions, represents the data volume exchanged between tasks.

3.2 Platform Formalization

The target computing environment is made up of a set of q heterogeneous compute resources completely interconnected. We assume that the communication between compute resources is performed without contention and can be overlapped with computation. We define two distinct matrices.

Computation_time matrix, with $v \times q$ dimensions, represents the execution time of tasks on compute resources. $Computation\_time_{i,\ j}$ is the estimated execution time of the task T_i on the compute resource PC_j.

The average execution time of a task T_i is:

$$\overline{Computation\_time_i} = \frac{\sum_{j=1}^{q} Computation\_time_{i,j}}{q} \tag{1}$$

Transfer_rate matrix, with $q \times q$ dimensions, represents the data transfer rate (bandwidth) between compute resources.

The communication time of an edge $(i, j) \in E$ in the DAG, which is the time taken to transfer data from task T_i (executed on PC_p) to task T_j (executed on PC_k), is defined as in the following:

$$Communication\_time_{i,j} = \frac{data_{i,j}}{Transfer\_rata_{p,k}} \tag{2}$$

When tasks T_i and T_j are executed on the same compute resource, we have $\text{Communication\_time}_{i,j}$ equal to zero. Consequently, the average communication time of an edge (i, j) is given in formula 3.

$$\overline{Communication\_time_{i,j}} = \frac{data_{i,j}}{\overline{Transfer\_rate}} \tag{3}$$

Where $\overline{Transfer\_rate}$ is the average of transfer rates between all the compute resources.

3.3 Makespan Formalization

To define the makespan, we use two attributes as defined in [7]:

1. *Earliest execution Start Time* (EST) of a task T_i assigned to a compute resource PC_j. EST is the earliest time during which a task T_i is started. As shown in the following,

the EST of a task T depends not only on the end of execution of the parent tasks of T but also on the data communication time, except when T is an entry task in which case EST is equal to zero.

$$\mathrm{EST}(T_{Entry}, PC_j) = 0 \tag{4}$$

$$\mathrm{EST}(T_i, PC_j) = \max\left\{avail[j], \max_{T_m \in pred(T_i)}(AFT(T_m) + Communication\_time_{m,i})\right\}. \tag{5}$$

Where:

- $pred(T_i)$ is the set of immediate parent tasks of T_i.
- $avail[j]$ is the earliest time at which the compute resource PC_j is available to execute a task, and
- AFT is the *Actual Finish Time* of a task as described here after.

2. *Earliest execution Finish Time* (EFT) of a task T_i on a compute resource PC_j corresponds to the time at which T_i ends its execution, that is, the starting time EST of T_i added to its execution time.

$$EFT(T_i, PC_j) = Computation\_time_{i,j} + EST(T_i, PC_j). \tag{6}$$

After task T_i is actually scheduled on the compute resource PC_j, *Actual Start Time* of the T_i is calculated as $AST(T_i) = EST(T_i, PC_j)$. In addition, *Actual Finish Time* of task T_i is defined as: $AFT\ (T_i) = EFT(T_i, PC_j)$.

When all the DAG tasks are scheduled, the completion time of the application is simply the AFT of the exit task.

$$\text{Makespan} = AFT(T_{exit}) \tag{7}$$

The objective of any scheduling algorithm is to find an assignment of tasks on the compute resources, that minimizes the makespan among other criteria. The next section deals with our contribution that aims to hybridize two heuristics in order to minimize makespan when scheduling tasks in a grid computing.

4 HEFT/DPSO* Hybridization

Our objective in this research work is twofold:

- First, we aim to adapt the basic DPSO [28] for the dependent tasks scheduling problem. In our proposed solution (DPSO*), after their assignment, tasks are ordered in such a way that the dependencies constraints are satisfied.

- Second, we aim to further improve the performances of our solution, i.e. the DPSO*, by combining it with the HEFT algorithm.

Our purpose here is to adapt the DPSO algorithm so that we can minimize the completion time of tasks when scheduled on a grid computing. Since the particles used by PSO algorithm are initialized randomly, our idea is to inject a particular particle initialized thanks to HEFT algorithm that is considered as a high-quality solution, in order to reduce significantly the convergence time. Before describing our proposed hybridization approach, we first recall the features of HEFT and describe our proposed DPSO* approach.

4.1 Heft

HEFT (Heterogeneous Earliest-Finish-Time) [7] is one of the most widespread scheduling-list algorithms. It determines a static scheduling of a DAG on a heterogeneous environment so as to minimize the makespan. As described in the following, HEFT has two execution steps.

1. **Task-prioritizing phase:** HEFT uses the *upward rank* attribute to order the tasks of the DAG. It is recursively defined by:

$$\mathrm{rank_u}(\mathrm{T_i}) = \overline{Computation\_time_i} + \max_{T_j \in succ(T_i)} (\overline{Communication\_time_{i,j}} + \mathrm{rank_u}(T_j)) \quad (8)$$

   Where $succ(T_i)$ is the set of immediate children of task T_i. The rank is calculated starting from the exit task.

$$\mathrm{rank_u}(\mathrm{T_{exit}}) = \overline{Computation\_time_{exit}}. \quad (9)$$

2. **Compute-resource selection phase:** Tasks are mapped to the adequate compute resources that minimize the EFT like in the formula 6.
   A variant of HEFT is the **Duplication based HEFT (DHEFT)** [29] that is based on task duplication. By duplicating dependent tasks and assigning them within the compute resources that host their children tasks, the communication overhead is reduced, hence improving the makespan of the application.

4.2 Adapted Discrete PSO (DPSO*)

PSO is an adaptive population-based search method inspired by social behavior patterns such as bird flocking and fish schooling. It can be implemented easily to solve various function optimization problems. Its main advantage is its fast convergence. Initially, PSO was used to solve continuous problems. However, a discrete binary version of PSO was introduced to solve discrete optimization problems in [30].

To solve scheduling problems, various versions of PSO were used like fuzzy PSO in [27] or Discrete PSO in [28]. DPSO deals with scheduling independent jobs in the grid environment. Since we are interested in this variant of PSO, we propose to adapt it to the dependent-tasks problem then hybridize our adapted DPSO, called DPSO*, with HEFT. Here, we explain its principle. In fact, our DPSO* algorithm initially generates randomly a swarm of particles. A particle is analogous to a bird flying through a search space. Each particle has a position X, a velocity V, and a fitness value. Particle's position is seen as a potential solution to the problem. Positions are evaluated by a fitness function to be optimized. Also, each particle knows its best past position it has reached (*pbest*) and the best position ever reached by any particle in the swarm (*gbest*). The movement of particles is influenced by its actual position and its velocity. Particle's velocity represents the direction and the magnitude of the next movement. It is calculated by considering its actual velocity, *pbest* and *gbest*. The next paragraphs discuss the features that characterize DPSO* algorithm.

Particle's Position. A particle's position represents a potential scheduling solution. We use the direct representation [28] to encode the scheduling solutions. The position (solution) is a vector X of v elements where v is the number of jobs. The elements of the vector are natural numbers included in range [0, q] where q is the number of compute resources in the grid. Hence, X[j] is the index of the computer resource where job j is assigned by the scheduler. For example, a solution of scheduling problem with 4 jobs to schedule and 2 available compute resources is represented by a vector of 4 elements X = (1, 0, 0, 1). So, jobs 1 and 4 are assigned to the compute resource indexed by 1 and jobs 2 and 3 are assigned to compute resource indexed by 0.

Particle's Velocity. The velocity is a $q \times v$ matrix called V where q is the number of available compute resources and v the number of jobs, as expressed in the following:

$$\mathrm{V[i,j]} \in [-v_{max}, V_{max}], \forall \mathrm{i} \in \{1, 2, \ldots, q\} \,\mathrm{et}\, \forall \mathrm{j} \in \{1, 2, \ldots, v\}$$

Initially, position's vectors and velocity's matrixes of particles are randomly generated as stated in Sect. 2.

Fitness Function. In general, the fitness function used to evaluate the particles is the makespan. Because makespan refers to the efficiency of the tasks-compute resources mapping, we have chosen to use it as a criterion to minimize.

$$\mathrm{Finess} = \mathrm{makespan}. \tag{10}$$

Movement. The movement is realized by firstly updating the matrix velocity and then the vector position of each particle. After each particle is moved, the *pbest* and *gbest* must be updated by checking the performance of each particle using the fitness function. The movement of the particles through the search space is described by the following algorithm.

Particles movement algorithm:

```
For each particle k = 1, ..., P do
    // Matrix velocity updating.
    For each job j = 1, ..., n do
       q =X_k^t[j];
       z =pbest_k^t[j];
       s = pbest_k^t[j];
       if q ≠z then
         V_k^t[q, j] =w.V_k^t[q, j] - c1   r1;
         V_k^t[z, j] w.V_k^t[z, j] + c1   r1;
       end if
       if q ≠s then
          V_k^t[q, j] =w.V_k^t[q, j] − c2   r2;
          V_k^t[s, j] =w.V_k^t[s, j] + c2   r2;
       end if
     end for

 // Vector position updating.
    For each job j = 1, ..., n do
     if V_k^t[ , j] = max {V_k^t[i, j]}∀i ∈ (1, 2, ..., m)then
       X_k^t[j]=  ;
     end if
    end for
end for
```

w, c1 and *c2* are the PSO parameters, *w* is the inertia weight, *c1* is the coefficient of the self-recognition component and *c2* is the coefficient of the social component, *r1* and *r2* are random numbers used to maintain the diversity of the swarm.

Dependency Constraint Supported. To take into consideration the dependency constraints between tasks, we characterize each task with the following parameters: task number, EST, EFT, and a tag value that indicates if the task is scheduled or not.

Before scheduling, the AST and AFT folders of each task are initially unknown, and all the tasks are tagged as not scheduled. Once DPSO* is applied, EST and EFT folders of each task are known and all the tasks are tagged as scheduled.

To get the final scheduling, our approach operates according to the following steps:

1. All the tasks are assigned to compute resources on which they will run. This step is similar to the basic DPSO that is suitable for independent tasks as described in [28]. However at this step, the order in which each task will start and finish its execution on a given compute resource is not known yet due to the task dependencies. Consequently, a second step is necessary as in the following.
2. In this second step, the start and finish execution time of each task will be defined on each compute resource. To do so, the DAG must be traveled downwards starting from the entry task. First, because entry task has no parent, it is tagged as scheduled and its EST is set to 0 (see formula 4) and its EFT is calculated using formula 5.

Second, the other tasks will wait until all their immediate parents are scheduled, in other terms, their ESTs and EFTs are calculated according respectively to the formulas 5 and 6.

DPSO* Algorithm. Before the start of the DPSO* execution some parameter values must be set. Then, particles are generated and initialized randomly. After that, they explore the search space trying to find a satisfactory solution for the problem until the maximum number of iteration is reached. A pseudo-code of the DPSO* algorithm is shown in the following.

4.3 The HEFT/DPSO* Hybridization

As depicted in Fig. 1, after generating the initial swarm, instead of randomly initializing all the particles, in our proposed DPSO* algorithm, one particle is initialized with the solution given by HEFT and other particles are randomly initialized. In this way, this step is optimized.

```
Initialize PSO parameter (swarm size, max_iter ,w , c1, c2)
Generate initial swarm
Initialize particles positions and velocities randomly
While (iter<max_iter) do
   For each particle k = 1, . . ., P do
     if Fitness (Xk) > Fitness (pbestk) then
       pbestk= Xk;
     end if
     if Fitness (pbestk) > Fitness (gbestk) then
       gbest=pbestk;
     end if
   end for
   For each particle k = 1, . . ., P do
     Movement of the particle k;
   end for
   iter =iter+1;
end while
```

5 Experimental Results

To evaluate the performance of our proposed algorithm, we have conducted some experiments and compared the resulting tests of our hybrid HEFT/DPSO* algorithm with HEFT and DPSO* algorithms regarding the makespan parameter. We have used a DAG generator called *RandomTaskGraphGenerator* to generate our DAGs that represent the applications to schedule.

The grid environment that we considered here is composed of several heterogeneous compute resources which are connected by heterogeneous links.

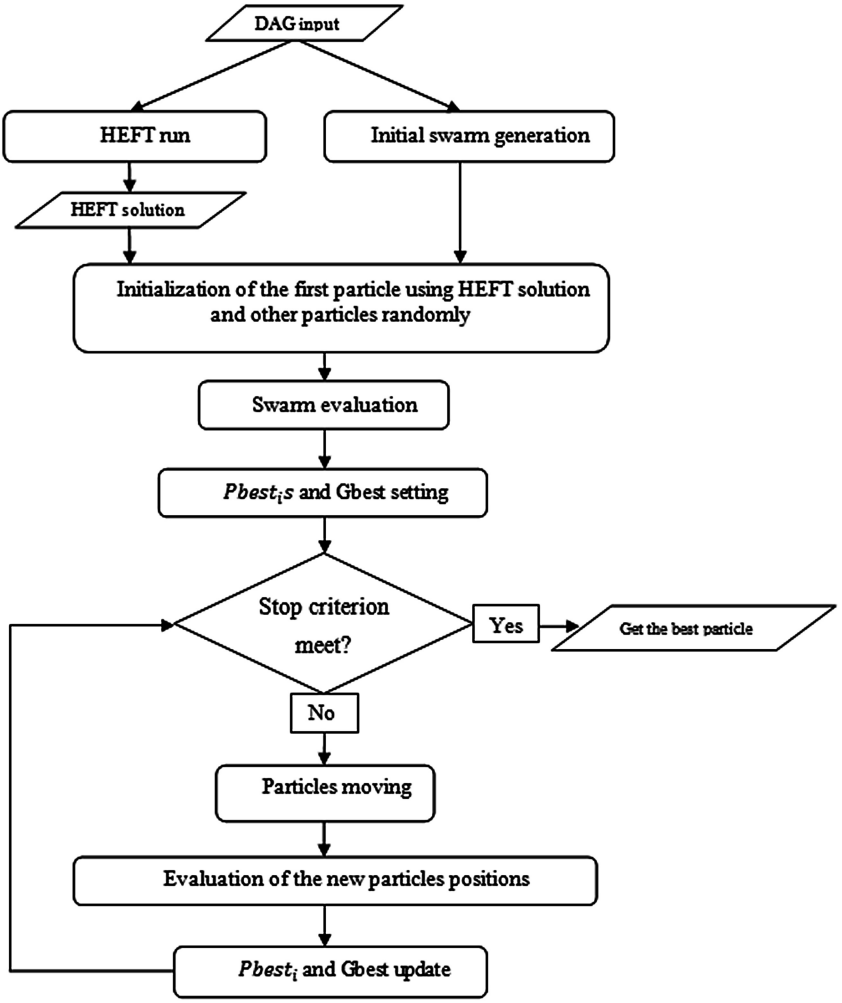

Fig. 1. Flow chart of HEFT/DPSO* hybridization

We assume that the computation time of each task on each compute resource, the data volume exchanged between tasks and the data transfer rate between compute resources are known.

For our simulation, the platform is described within a configuration file that contains the following information: the number of tasks composing each application, the number of compute resources of the platform and three matrices. The first is the computational cost matrix with "Number of tasks × Number of compute resources" dimension (line index represents the number of tasks and column index represents the number of compute resources). The second is the data transfer speed matrix with "Number of compute resources × Number of compute resources" dimension (line and column indexes represent compute resource numbers). The value '0' means there is no

transfer between a compute resource and another; besides the transfer speed between a compute resource and itself is null. The third one is the data matrix that contains the data transferred between tasks. This matrix has "Number of tasks × Number of tasks" dimension (the line and column indexes represent task numbers). The value '−1' means that there is no data exchanged between tasks. A value '0' means that a task has another kind of dependence, other than data transfer, with another task. Concerning the positive values of the data matrix, each value corresponds to the volume of data transfer between the two corresponding tasks.

Based on the configuration file, we conducted our measurements using five applications with different number of tasks on a grid environment with different number of available compute resources.

Since the results of our DPSO* and our hybrid algorithm are stochastic (due to the DPSO behavior), we repeated each experiment 10 times and recorded the makespan value of the best solution obtained. For HEFT, it is executed only once since it is a deterministic algorithm.

Specific parameter settings used by our hybrid HEFT/DPSO* and our DPSO* are described in Table 1.

In our experiments, we measured the makespan criterion by varying the number of compute resources in one side, and the number of tasks in the other side.

Table 1. Parameter settings of DPSO.

DPSO* parameter	Value
Size of swarm	50
Maximum iteration	1000
Self-recognition coefficient c1	2
Social coefficient c2	1

5.1 First Performance Study

We used an application of 40 tasks which we run on a grid environment with different number of available compute resources (2, 3, 5, 7 and 10), and we measured the makespan as in Table 2.

Table 2. Makespans comparison according to the number of compute resources.

Compute resource number	HEFT/DPSO*	DPSO*	HEFT
2	733	736	922
3	560	580	620
5	425	437	441
7	383	392	399
10	296	329	308

Figure 2 shows the makespan as measured for each scheduling algorithm when varying the number of compute resources. According to these measurements, we notice that in respect to makespan, our hybrid HEFT/DPSO* algorithm offers better results than DPSO* and HEFT algorithms.

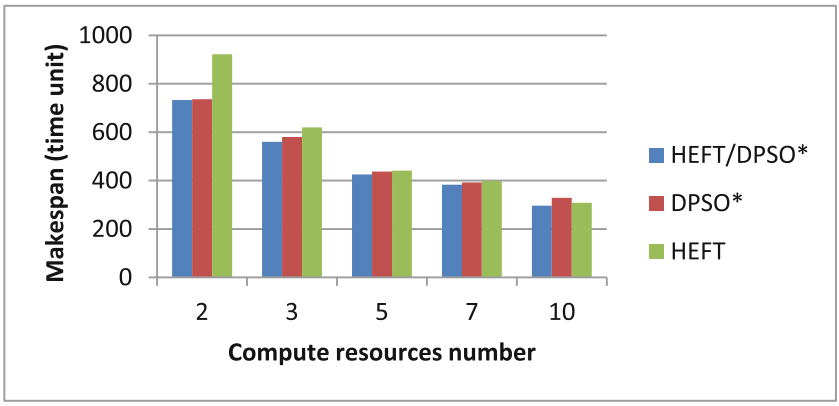

Fig. 2. Makespan comparison according to the number of compute resources

5.2 Second Performance Study

To measure the makespan criterion when varying the number of tasks, we considered five applications with different number of tasks (10, 20, 40, 60 and 100) that we run on a grid environment with 5 available compute resources. We compared the results of our hybrid solution with those obtained with the following scheduling algorithms: DPSO* and HEFT. Table 3 shows the values of the makespan as obtained in the different situations.

Table 3. Makespan comparison according to the number of tasks.

Task number	HEFT/DPSO*	DPSO*	HEFT
10	71	71	84
20	225	228	299
40	431	411	441
60	446	511	654
80	434	573	523
100	681	849	701

Figure 3 depicts the makespan comparison between different scheduling algorithms when varying the number of tasks. Even when varying the number of tasks, the makespan seems to be better especially when the number of tasks is relatively important.

According to Figs. 2 and 3, we can notice that the performances of the hybrid HEFT/DPSO* solution and those of the DPSO* are better than HEFT heuristic regarding the makespan criterion. Moreover, in the majority of cases, the hybrid solution provides better makespans than the DPSO*.

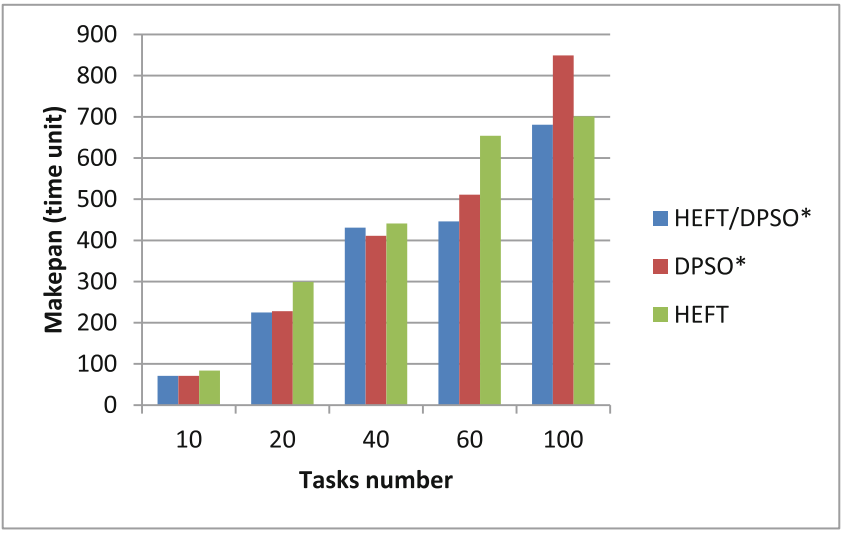

Fig. 3. Makespan comparison according to the number of tasks

6 Conclusion and Future Work

In this paper, a dependent task scheduling algorithm for computational grid has been proposed based on hybridization of two heuristics. The first one is a list-scheduling heuristic which is the well-known HEFT used to schedule DAGs. The second one is based on a meta-heuristic called Particle Swarm Optimization (PSO). A discrete version of PSO has been adapted to handle the scheduling of DAGs. Our objective was to minimize the makespan of applications that are executed on a grid environment. However, we plan to measure other criteria like energy consuming in our future work.

We showed in this article that our proposed scheduling approach gives better results in term of completion time than HEFT and our DPSO*.

At last, we aim to complete our current research work by hybridizing the DHEFT algorithm with a PSO technique to support the duplication of tasks in one side, and in the other side to measure the impact of the clustering approach that groups inter-dependent tasks into meta-tasks by using a genetic algorithm and then by scheduling them using HEFT algorithm.

References

1. Cafaro, M., Aloisio, G.: Grids, Clouds, and Virtualization. 1st edn., Spring (2011). ISBN 978-0-85729-049-6
2. Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical report No. 2006-504. School of Computing, Queen's University, Kingston, Ontario

3. Casavant, T., Kuhl, J.: A taxonomie of scheduling in general-purpose distributed computing systems. IEEE Trans. Softw. Eng. **14**(2), 141–154 (1988)
4. Braun, R., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B., Hensgen, D., Freund, R.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
5. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
6. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. In: Xhafa, F., Abraham, A. (eds.) Metaheuristics for Scheduling in Distributed Computing Environments. Studies in Computational Intelligence, vol. 146, pp. 173–214. Springer, Heidelberg (2008)
7. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
8. Radulescu, A., van Gemund, A.J.C.: On the complexity of list scheduling algorithms for distributed-memory systems. In: Technical report No. 1-68340-44(1999)02, January 1999
9. Kwok, Y., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to muliprocessors. IEEE Trans. Parallel Distrib. Syst. **7**(5), 506–521 (1996)
10. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. **4**(2), 75–87 (1993)
11. Ma, T., Buyya, R.: Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids. In: IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005) (2005)
12. Yang, T., Gerasoulis, A.: DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Trans. Parallel Distrib. Syst. **5**(9), 951–967 (1994)
13. Liou, J., Palis, M.A.: An efficient clustering heuristic for scheduling DAGs on multiprocessors. In: Proceedings of the Symposium Parallel and Distributed Processing (1996)
14. Boeres, C., Filho, J.V., Rebello, V.E.F: A cluster-based strategy for scheduling task on heterogeneous processors. In: IEEE Symposium on Computer Architecture and High Performance Computing, pp. 214–221, October 2004
15. Kruatrachue, B., Lewis, T.: Grain size determination for parallel processing. IEEE Softw. **5**, 23–32 (1988)
16. Ahmad, I., Kwok, Y.-K.: A new approach to scheduling parallel programs using task duplication. In: IEEE International Conference on Parallel Processing, vol. 2 (1994)
17. Darbha, S., Agrawal, D.P.: Optimal scheduling algorithm for distributed-memory machines. IEEE Trans. Parallel Distrib. Syst. **9**(1), 87–95 (1998)
18. Chung, Y.-C., Ranka, S.: Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed-memory multiprocessors. In: Proceedings of the Supercomputing, pp. 512–521 (1992)
19. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. **15**(2), 107–118 (2004)
20. Wang, L., Siegel, H.J., Roychowdhury, V.P., Maciejewski, A.A.: Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. J. Parallel Distrib. Comput. **47**(1), 8–22 (1997)
21. Martino, V.D., Mililotti, M.: Sub optimal scheduling in a grid using genetic algorithms. Parallel Comput. **30**, 553–565 (2004)

22. Gao, Y., Rong, H., Huang, J.Z.: Adaptive grid job scheduling with genetic algorithms. Future Gener. Comput. Syst. **2**, 151–161 (2005)
23. Aggarwal, M., Kent, R.D., Ngom, A.: Genetic algorithm based scheduler for computational grids. In: Proceedings of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS 2005), May 2005
24. Song, S., Kwok, Y., Hwang, K.: Security-driven heuristics and a fast genetic algorithm for trusted grid job scheduling. In: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005), April 2005
25. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ (1995 in press)
26. Zhang, L., Chen, Y., Sun, R., Jing, S., Yang, B.: A task scheduling algorithm based on PSO for grid computing. Int. J. Comput. Intell. Res. **4**(1), 37–43 (2008)
27. Liu, H., Abraham, A., Hassanien, A.E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst. **26**, 1336–1343 (2010)
28. Izakian, H., Ladani, B.T., Abraham, A., Snasel, V.: A discrete particle swarm optimization approach for grid job scheduling. Int. J. Innovative Comput. Inf. Control **6**(9), 4219–4233 (2010)
29. Zhang, Y.-Y., Inoguchi, Y., Shen, H.: A dynamic task scheduling algorithm for grid computing system. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, pp. 578–583. Springer, Heidelberg (2004)
30. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 5 (1997)