A Static Tasks Assignment For Grid Computing

Meriem Meddeber
Department of Mathematic and Computer Science
University of Mascara
Algeria, 29000
Email: m.meddeber@yahoo.fr

Belabbas Yagoubi
Department of Computer Science
University of Oran
Algeria, 31000
Email: byagoubi@gmail.com

Walid Kadri
Department of Computer Science
University of Oran
Algeria, 31000
Email: walidk.04@hotmail.com

***Abstract*—Tasks assignment in grid computing is a challenge for most researchers and developers of these types of systems. This is due to characteristics inherent to grid infrastructures, namely, *heterogeneity*, *dynamicity* and *scalability*. It becomes much more complex when it comes to assign tasks with precedence constraints represented by a Directed Acyclic Graph. The task assignment problem is well known to be *NP-complete*.**
In this paper, we present and evaluate a dependent task assignment strategy for Grids. Our goal is two folds: first we reduce , whenever possible, the average response time of tasks submitted to the grid, and secondly, we reduce the transfer cost inducing by the tasks transfer respecting the dependency constraints.

Index Terms*—Dependent Tasks; Grid Computing; Tasks Assignment; Directed Acyclic Graph; Grid Model; Gridsim

I. Introduction

Computational Grids [1] are emerging as next generation parallel and distributed computing platforms for solving large-scale computational and data-intensive problems in science, engineering, and commerce. They enable the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, databases, data sources, and specialized devices owned by different organizations.
Task assignment is an important issue in grid computing systems, which provides a better exploitation of the system parallelism and improves its performance. The so called task assignment problem is a combinatorial optimization problem which consists of assigning a given computer program formed by a number of tasks to a number of processors/machines, subject to a set of constraints, and in such a way a given cost function to be minimized. The constraints of the task assignment problem are usually related to the resources available for the processors in the system. The cost function for a task assignment problem usually involves the minimization of completion time of the entire program, the minimization of the communication time among tasks or the minimization or the processors load [2].
In general, scheduling applications in a distributed system is a NP-hard problem even when the tasks are independent. The problem is much more difficult when the tasks have dependencies because the order of task execution as well as task-machine pairing affects overall completion time[3].
Tasks assignment policies for distributed systems can be generally categorized into static tasks assignment policies and dynamic tasks assignment policies [4] :

- Static tasks assignment policies use some simple system information, such as the various information related to average operation, operation cycle, and etc., and according to these data, tasks are distributed through mathematical formulas or other adjustment methods, so that every node in the distributed system can process the assigned tasks until completed. The merit of this method is that system information is not required to be collected at all times, and through a simple process, the system can run with simple analysis. However, some of the nodes have low utilization rates. Due to the fact that it does not dynamically adjust with the system information, there is a certain degree of burden on system performance.
- Dynamic tasks assignment policies refer to the current state of the system or the most recent state at the system time, to decide how to assign tasks to each node in a distributed system. If any node in the system is over-loaded, the over-loading task will be transferred to other nodes and processed, in order to achieve the goal of a dynamic assignment. However, the migration of tasks will incur extra overhead to the system. It is because the system has to reserve some resources for collecting and maintaining the information of system states. If this overhead can be controlled and limited to a certain acceptable range, in most conditions, dynamic tasks assignment policies out perform the static tasks assignment policies.

In this paper, we will form a set of clusters from an initial collection of nodes using a distributed clustering algorithm noted *DCA*. Each clusterhead will use a static assignment strategy to place an application modeled by a dependent task graph on a heterogeneous computing platform. This algorithm is called HEFT (Heterogeneous Earliest FinishTime).
The rest of this paper is organized as follows. In section 2, we present the tasks assignment algorithms. Section 3, describes Tasks assignment in Grid computing environments. A Preliminaries and problem definition is done in Section 4. In section 5, we will describes the main steps of the proposed assignment strategy. We evaluate the performance of the proposed scheme in Section 6. Finally, Section 7 concludes the paper.

II. Task assignment algorithms

Assignment algorithms focus on the number of factors to be considered, when scheduling resources to tasks. Some algorithms consider a single factor, while others consider multiple factors. Different algorithms are suitable for different applications. Consequently, each characteristic and suitable domain should be considered in selecting a proper scheduling algorithm [11], [12].

A. Single factor assignment algorithm

In this method, only a single factor is considered, such as node resource or task execution demand. Based upon this knowledge, this assignment method is simpler and does not increase the system load in practical applications. Because only one factor is considered, this method is suitable for pure and stable environments. It is not suitable for complex and multi-variant environments like grids.

1) First come first served assignment algorithm: First come first served is the simplest form of a assignment algorithm. It is based on the theory that resources are assigned to the primary proposed tasks. After the first task is completed, the resource is re-assigned to the next proposed task. This method uses the order of executed tasks in the order of submission. If a significant amount of time is required to complete the first task, the same amount of time is spent waiting for the execution of the next task. Therefore, a convoy effect is created and the entire system performance will be reduced.

2) Priority-assignment algorithm: Priority-assignment algorithms give priority to the order of task execution. If more than two tasks have the same priority, then the First come first served method will be applied. Since the order of execution is defined by the order of priority, the priority order decision is the biggest problem in the priority-assignment algorithm. If the priority order decision is incorrect, then the resources will be continuously occupied by a high priority task, causing unlimited deadlocks or starvation.

B. Multiple factors assignment algorithm

The characteristics of this approach are the significance of simultaneous considerations of nodes and resource demand loads. Although multiple factors concurrently considered and tasks are efficiently completed, a greater load will be placed on the system. Therefore, this approach is more suitable for complex and volatile environments.

1) Opportunistic Load Balancing: assigns each job in arbitrary order to the processor with the shortest schedule, irrespective of the expected time to compute on that processor. Opportunistic Load Balancing is intended to try to balance the processors, but because it does not take execution times into account it finds rather poor solutions.

2) Minimum completion time : Minimum completion time assigns each task, to the node, in an arbitrary order with the minimum expected completion time for that task. This method causes assignment of some tasks to nodes that do not have the minimum execution time for that task.

3) MinMin: Minmin establishes the minimum completion time for every unscheduled task. It then assigns the task to the node that offers the minimum completion time. Minmin uses the same intuition as Minimum completion time algorithm, but since it considers the minimum completion time for all tasks; it can schedule at each iteration, the task that will least increase the overall make-span, to help balance the nodes better than Minimum completion time algorithm.

4) Max-min: Max-min is very similar to Min-min. Again the minimum completion time for each job is established, but the job with the maximum minimum completion time is assigned to the corresponding processor.

III. Tasks Assignment in Grid Computing Environments

Tasks assignment systems for Traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Tasks assignment in Grid environments is significantly complicated by the unique characteristics of Grids:

- *Heterogeneity of the grid resources*
  Heterogeneity exists in two categories of resources. First, networks used to interconnect these computational resources may differ significantly in terms of their bandwidth and communication protocols. Second, computational resources may have different hardware, computer architectures, physical memory size, CPU speed and so on and also different software, such as different operating systems, cluster management software and so on. This characteristic complicates the system workload estimation because the heterogeneous resources could not be considered uniformly.
- *Grid resources are dynamic*
  In traditional distributed systems, such as a cluster, the pool of resources is assumed to be fixed or stable. In the Grid, this character is not verified because of computational resources and communication networks dynamicity. Both computational resources availability and capability will exhibit dynamic behaviour. On one hand new resources may join the Grid and on the other hand, some resources may become unavailable due do problems such as network failure. This pose constraints on applications such as fault tolerance. A resource that connects or disconnects must be detected and taken into account by the system.

IV. Preliminaries and problem definition

A. Grid Model

We model a grid (Fig. 1 (a)) by an undirected graph *G = (V,E)* in which V , $|V|$ = n, is the set of nodes and there is an edge $\{u,\ v\}$ $\in$ E if and only if u and v can mutually receive each others' transmission (this implies that all the links between the nodes are bidirectional). Due to the dynamicity of the grid, the graph can change in time.
Every node v in the network is assigned a unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v. Finally, we consider weighted networks, i.e., a weight w_v (a real number $\geq$ 0) is assigned to each node v $\in$ V of the network. In this paper The width correspond to the MIPS of the node.
Clustering [5] a network means partitioning its nodes into clusters, each one with a clusterhead and (possibly) some ordinary nodes. The choice of the clusterheads is here based on the weight associated to each node: the bigger the weight of a node, the better that node for the role of clusterhead. In order to meet the requirements imposed by the wireless, mobile nature of these networks, a clustering algorithm is required to partition the nodes of the network so that the following clustering properties are satisfied:

1) Every ordinary node has at least a clusterhead as neighbor (dominance property).
2) Every ordinary node affiliates with the neighboring clusterhead that has the bigger weight.
3) No two clusterheads can be neighbors (independence property).

The DCA algorithm is required to be executed at each node (i.e., the algorithm should be distributed) with the sole knowledge of the topology local to each node. Fig 1. (b) illustrates a correct clustering for the ad hoc network of Fig. 1 (a) (the clusterheads of each cluster are the squared nodes).

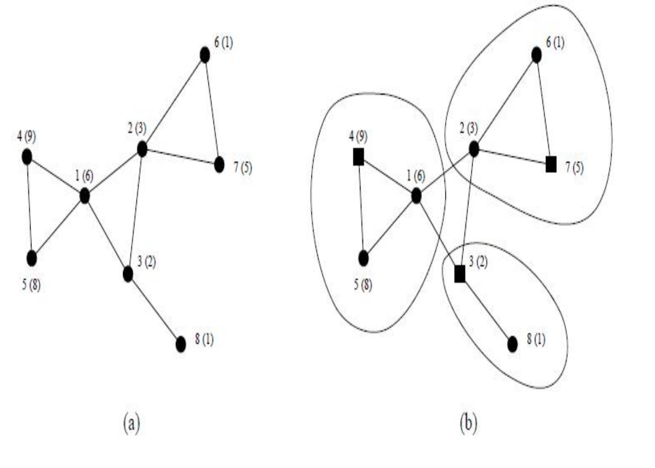

Fig. 1. (a) A network G with nodes v and their weights (w_v), 1 $\leq$ v $\leq$ 8, and (b) a correct clustering for G.

B. Tasks Model

Before explaining the proposed strategy, some definitions be given in the following paragraph.

1) Definition 1 (Task graph): An application can be represented by a directed acyclic graph (*DAG*) *D= (V, E)*, where *V* is a set of *v* nodes and *E* is a set of directed *e* edges. A node in the DAG represents a task which in turn is a set of instructions which must be executed sequentially without preemption in the same processor. The edges in the DAG, each of which is denoted by (n_i, n_j), correspond to the precedence constraints among the nodes. The weight of an edge is called the communication cost of the edge and is denoted by C_{ij}. The source node of an edge is called the *parent node* while the sink node is called the *child node*. A node with no parent is called an *entry node* and a node with no child is called an *exit node*[6].
Figure2, shows a task precedence graph constituted by five tasks, with one entry task $T_1$and three exit tasks T_3,T_4,T_5.

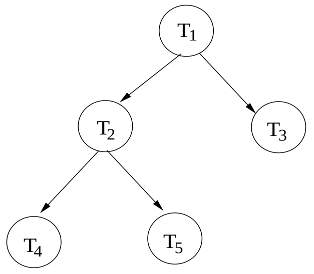

Fig. 2. Directed Acyclic Graph

The objective of DAG scheduling is to minimize the overall program finish-time by proper allocation of the tasks to the processors and arrangement of execution sequencing of the tasks. Scheduling is done in such a manner that the precedence constraints among the program tasks are preserved. The overall finish-time of a parallel program is commonly called the schedule length or makespan. Some variations to this goal have been suggested. For example, some researchers proposed algorithms to minimize the mean flow-time or mean finish-time, which is the average of the finish-times of all the program tasks [7].

2) Definition 2 (Tasks properties): We consider that tasks arrive randomly with a random computation length, an arrival time and precedence constraints. In our work, we generate randomly precedence constraints between tasks. Also, we believe that tasks can be executed on any computing element and each *CE* can only execute one task at each time point, the execution of a task cannot be interrupted or moved to another CE during execution.
We also assume that a task cannot start execution before it gathers all of the messages from its parent tasks. The communication cost between two tasks assigned to the same processor is supposed to be zero.

V. Proposed Strategy

Step 1 : Clustering

Each node execute the DCA algorithm (proposed by [5]) with the sole knowledge of the executing node's unique identifier, ID, its weight, and the IDs and the weights of its neighbors (namely, the algorithm is distributed and relies only on local information).

DCA (Fig. 3) algorithm use only two types of messages:

1) *Ch(v)*: used by a node v to make its neighbors aware that it is going to be a clusterhead,
2) *Join(v; u)*: with which a node v communicates to its neighbors that it will be part of the cluster whose clusterhead is node u.

Every node starts the execution of the algorithm at the same time, running the procedure Init. Only those nodes that have the biggest weight among all the nodes in their neighborhood will send a Ch message (init nodes).
All the other nodes just wait to receive a message.

- On receiving a Ch message from a neighbor u, node v checks if it has received from all its neighbors z such that $w_z > w_u$, a Join(z; x) message. In this case, v will not receive a Ch message from these z, and u is the node with the biggest weight in v's neighborhood that has sent a Ch message. Thus, v joins u, and quits the algorithm execution (it already knows the cluster to which it belongs, i.e., its clusterhead). If there is still at least a node z, $w_z > w_u$, that has not sent a message yet, node v records that u sent a Ch message, and keeps waiting for a message from z.
- On receiving a Join(u; t) message, node v checks if it has previously sent a Ch message. If this is the case, it checks if node u wants to join v's cluster (v = t). Then, if all v's neighbors z such that wz ¡ wv have communicated their willingness to join a cluster, v quits the execution of the DCA. Notice that, in this case, node v does not care about its neighbors y (if any) such that $w_y > w_v$, because these nodes have surely joined a node x such that $w_x > w_v$ (thus permitting v to be a clusterhead). If node v has not sent a Ch message, before deciding what its role is going to be, it needs to know what all the nodes z such that $w_z > w_v$ have decided for themselves. If v has received a message from all such nodes, then it checks the nature of the messages received. If they are all Join messages, this means that all those neighbors z have decided to join a cluster as ordinary nodes. This implies that now v is the node with the biggest weight among the nodes (if any) that have still to decide what to do. In this case, v will be a clusterhead. At this point, v also checks if each neighbor y such that $w_y < w_v$ has already joined another cluster. If this is the case, v quits the algorithm execution: it will be the clusterhead of a cluster with a single node. Alternatively, if v has received at least a Ch message from z, then it joins the cluster of the neighbor with the biggest weight that sent a Ch message, and quits the execution of the DCA.

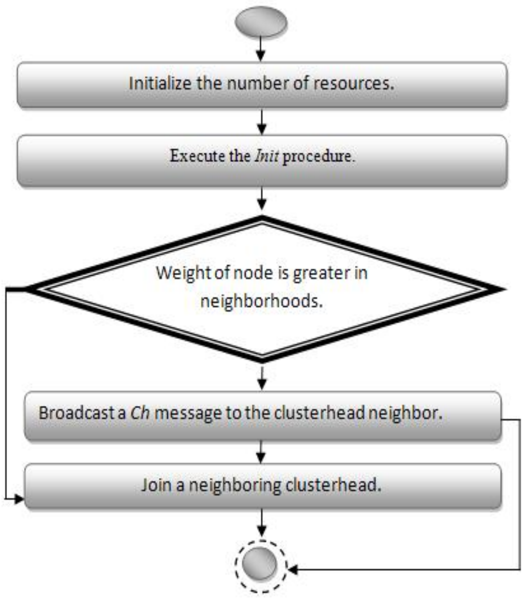
Initialize the number of resources.
Execute the *Init* procedure.
Weight of node is greater in neighborhoods.
Broadcast a *Ch* message to the clusterhead neighbor.
Join a neighboring clusterhead.

Fig. 3. Distributed Clustering Algorithm

Step 2 : Tasks Scheduling

Each clusterhead will use a static assignment strategy to place an application modeled by a dependent task graph on a heterogeneous computing platform. This algorithm is called HEFT (Heterogeneous Earliest FinishTime) and was proposed by [8].
Heterogeneous Earliest Finish Time heuristic has been one of the most often cited and used, having the advantage of simplicity and producing generally good schedules with a short makespan. HEFT is essentially a list scheduling heuristic that constructs first a priority list of tasks and then makes locally optimal allocation decisions for each task on the basis of the tasks estimated finish time [9].
The HEFT (Fig. 3) algorithm consists of 3 phases [10]:

1) *Weighting*: it assigns weights to the nodes and edges in the graph;
2) *Ranking*: it creates a sorted list of tasks, ordered by their priorities;
3) *Mapping*: it assigns tasks to resources.

In *phase 1*, the weights assigned to nodes correspond to the predicted execution times of the tasks, while the edge weights correspond to the predicted data transfer times between the resources. HEFT assumes these times to be known. In environments with homogeneous resources, the weights directly reflect the predicted times. In heterogeneous environments, the weights must be adjusted considering variances in execution times on different resources, and different data transfer times on data links. Several adjustment methods were proposed and compared. Each of them provides another accuracy with respect to the considered scenario. The common method is to take the arithmetic average over all resources.

In the ranking *phase 2*, the workflow graph is traversed backward, and a rank value is assigned to each of the tasks. The rank value denotes the tasks priority, thus a higher rank means a greater priority. The rank of a task is equal to the tasks weight plus the maximum successive weight. This means for every edge leaving the task, that the edge weight is added to the previously calculated rank of the adjacent node, and that the maximum of the summations is chosen. In the end, the tasks are sorted by decreasing rank order. This results in an ordered ranking list.

In the *mapping phase*, tasks from the ranking list are mapped to the resources one after the other, and each task is assigned to that resource which minimizes the tasks earliest expected finish time.

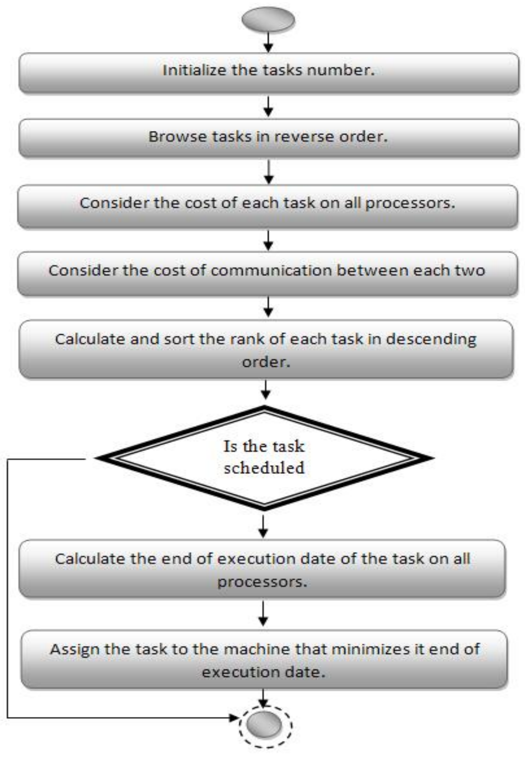
Initialize the tasks number.
Browse tasks in reverse order.
Consider the cost of each task on all processors.
Consider the cost of communication between each two
Calculate and sort the rank of each task in descending order.
Is the task scheduled
Calculate the end of execution date of the task on all processors.
Assign the task to the machine that minimizes it end of execution date.

Fig. 4. Heterogeneous Earliest FinishTime

VI. Experimental Study

To test and evaluate the performance of our model, we developed our strategy under the GridSim[13] simulator. In GridSim, application tasks/jobs are modelled as Gridlet objects that contain all the information related to the job and the execution management details, such as:

1) *Resources parameters* These parameters give information about computing elements, clusters and networks. A node is characterized by its capacity, speed and networks bandwidth sizes.
2) *Tasks parameters* These parameters include the number of tasks queued at every node, task submission date, number of instructions per task, cumulative processing time, cumulative waiting time and so on.

As performance measures, we are interested in average response time of tasks.
To obtain results that are as consistent as possible, we repeated the same experiments more than ten 5 times.
All these experiments were performed on a PC 1.7 GHz Pentium IV, with 1GB of memory and running on Windows XP.

The tasks sizes was generated randomly between 1000 and 200000 MI (Million of Instructions). For each node we generate randomly associated speed varying between 5 and 30 MIPS.
The results presented are for 400 nodes.

We compared the performance of our strategy with:

1) *Random Strategy :* Tasks are affected randomly to resources.

2) *Heft Strategy :* Tasks are affected using HEFT algorithm, without use of DCA algorithm.

3) *Strategy based on meta-tasks :* The initial precedence graph is divided into several graphs, including each one a number of dependent tasks called meta-tasks. note that there is no precedence relations between the meta-tasks[14].

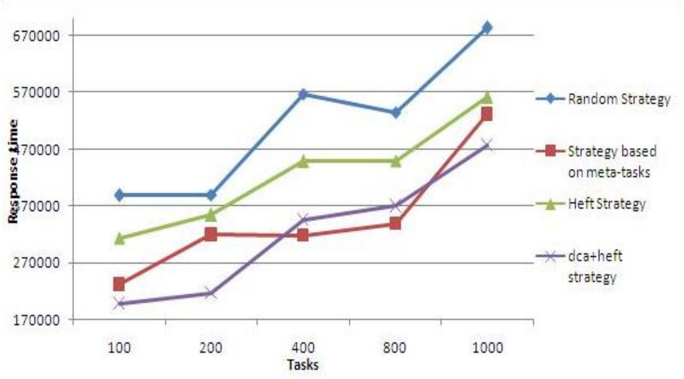

Fig. 5. Experimental Results

We note that :

- The Random strategy gives the worst results.
- The heft Strategy gives better results than Random strategy.
- Our strategy is better than the strategy based on meta-tasks except the cases of 400 and 800 nodes. This leads to the following conclusion: if the grid is stable, the strategy based on meta-tasks is favorable, in other cases it is preferable to use the proposed strategy.
- By increasing the number of tasks, the gains of the proposed approach increases compared to the random strategy.

VII. Conclusion

Grid computing architectures are developing rapidly. They are hardware and software infrastructures that provide dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. This technology is a type of distributed system which supports the sharing and coordinated use of resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal.

Grid computing have the potential to provide low cost and high-performance computing whenever computational applications can be broken into tasks that can be distributed to the various machines for parallel execution. A grid computing system has potential advantages over homogeneous systems because some tasks run faster on one type of machine while other types of tasks may run faster on a different machine.

Tasks assignment systems for Traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Tasks assignment in Grid environments is significantly complicated by the unique characteristics of Grids : Heterogeneity of the grid resources and Grid resources are dynamic.

In this paper, we formed a set of clusters from an initial collection of nodes using a distributed clustering algorithm noted *DCA*. Each clusterhead will use a static assignment strategy to place an application modeled by a dependent task graph on a heterogeneous computing platform. This algorithm is called HEFT (Heterogeneous Earliest FinishTime).

To test and evaluate the performance of our model, we developed our strategy under the GridSim simulator written in Java. We have randomly generated nodes with different characteristics and a set of dependent tasks.
The first experimental results are encouraging since we can significantly reduce the average response time

As future work, we plan to extend this strategy and test it on other existing simulators of grid. we want also to improve the proposed strategy by integrating the multi-agent systems.

Acknowledgment

The authors would like to thank N. Douaibia, and B. Dougani for their contribution to the implementation of this work.

References

[1] Shupeng Wang, Xiaochun Yun and Xiangzhan Yu, *Survivability-based Scheduling Algorithm for Bag-of-Tasks Applications with Deadline Constraints on Grids*, International Journal of Computer Science and Network Security, **6**: 4, 2006.

[2] Sancho Salcedo-Sanza,Yong Xub and XinYaob, *Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems*, Computers and Operations Research, **33** : 820835, 2006.

[3] Wayne F. Boyer and Gurdeep S. Hura, *Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments*, J. Parallel Distrib. Comput. **65**: 1035-1046, 2005.

[4] K. Q. Yan, S. C. Wang, C. P. Chang and J. S. Lin, *A hybrid load balancing policy underlying grid computing environment*, Computer Standards and Interfaces. **29**, 161-173, 2007.

[5] Stefano Basagni, *Distributed Clustering for Ad Hoc Networks*, in Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, June 1999.

[6] Yu-Kwong Kwok, Ishfaq Ahmad, *Static scheduling algorithms for allocating directed task graphs to multiprocessors*, ACM Computing Surveys. **31**: 406-471,1999.

[7] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surveys, **31** : 406471, 1999.

[8] H. Topcuoglu, S. Hariri and Min-You-wu, *Peformance-effective and low-complexity task scheduling for heterogeneous computing*, IEEE Transactions on Parallel and Distributed Systems, Vol 13, Issue 3, pp. 260-274, March 2002.

[9] Luiz F. Bittencourt , Rizos Sakellariou , Edmundo R. M. Madeira, *DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm*, Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, p.27-34, February 17-19, 2010.

[10] Dietmar Sommerfeld and Harald Richter, *A Novel Approach to Workflow Scheduling in MediGRID*, Technical Report Series, Department Of Informatics,Clausthal University of Technogy.

[11] Kuo-Qin Yana,Shun-Sheng Wanga, Shu-Ching Wang, and Chiu-Ping Changa, *Towards a hybrid load balancing policy in grid computing system*, Pergamon Press, Inc. Tarrytown, NY, USA, 2009.

[12] Graham Ritchie and John Levine, *A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments*, In 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.

[13] R. Buyya and M. Murshed, *Gridsim : A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*, The Journal of Concurrency and Computation: Practice and Experience (CCPE). **14**: 13-15, 2002.

[14] Meriem Meddeber, Belabbes Yagoubi. *Equilibrage de Charge pour les Grilles de Calcul: Classe des Tches Dpendantes et Indpendantes*. Confrence Internationale sur l'Informatique et ses Applications (CIIA), Saida, Algrie, 3,4 Mai 2009.