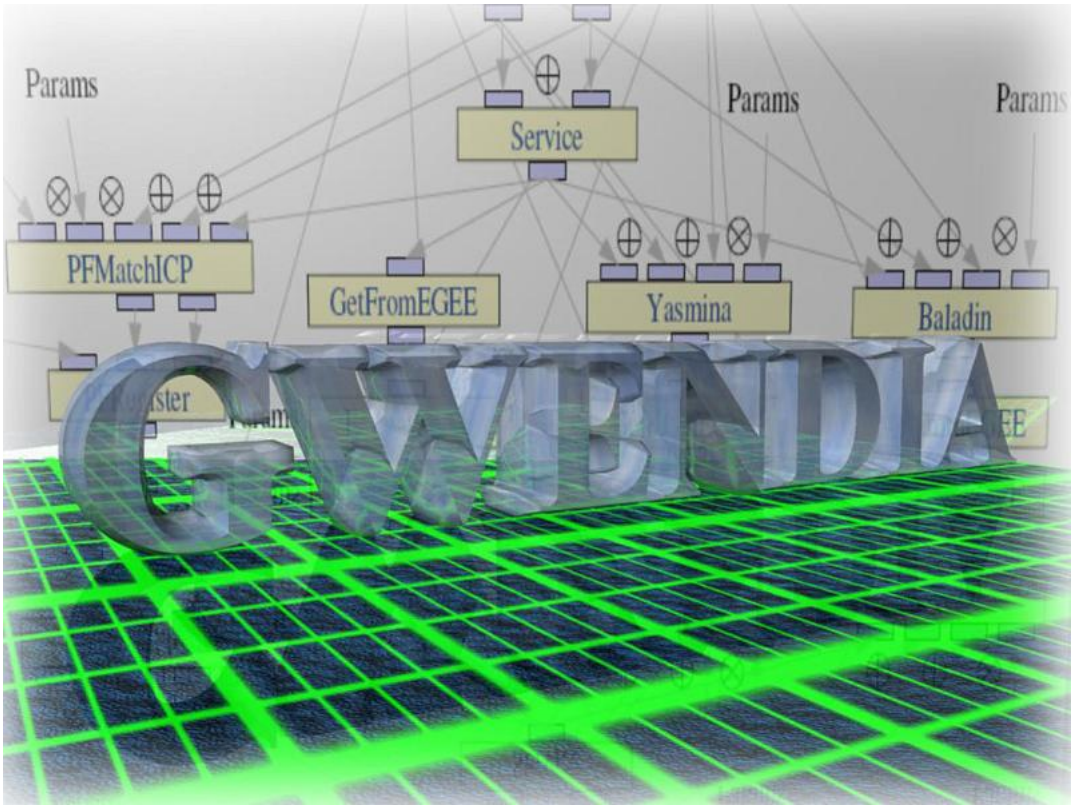
Gwendia ANR-06-MDCA-009
Grid Workflow Efficient Enactment for Data Intensive Applications

L3.3: Evaluation of Online Multi-Workflow Heuristics based on List-Scheduling Algorithms



Raphaël BOLZE GRAAL (LIP) raphael.bolze@ens-lyon.fr
Frédéric DESPREZ GRAAL (LIP) frederic.desprez@ens-lyon.fr
Benjamin Isnard GRAAL (LIP) benjamin.isnard@ens-lyon.fr

Abstract

This deliverable addresses the online scheduling of several applications modeled as workflows in grids, and proposes different heuristics to tackle this problem

Contents

1 Abstract

This deliverable addresses the online scheduling of several applications modeled as workflows in grids. Although a large amount of scheduling heuristics in grids have been proposed in the literature, most of them target only single tasks graph scheduling.

We propose to study this problem by extending a well-known list scheduling heuristic (HEFT) and adapt it to the multi-workflow context. We have designed six different heuristics based on HEFT key ideas. Those heuristics have been implemented into a grid middleware named DIET which implements the GridRPC API from the OGF. A new component called MA_{DAG} is introduced to take into account multiple submissions of tasks graphs.

We validated and compared our heuristics behavior with case study applications taken from a bioinformatics project. It appears in our given scenario that all of proposed heuristics have similar performance in terms of global makespan which is near the theoretical lower bound. However there exist significant differences comparing the slowdown of each application type.

2 Introduction

Grid computing has gained more and more importance because of the computing power and storage capacity needed by scientific applications. During several years, grid computing was reserved to computer scientists that developed ad-hoc solutions to manage and address the computing power of grid platforms. The development of several middlewares made the access to production grids across the world easier and allowed to tackle more complicated applications.

In many scientific areas, such as high-energy physics, bioinformatics, astronomy, and others, we encounter applications involving numerous simpler components that process data sets, execute scientific computations, and share both data and computing resources. Such applications consist of multiple components (tasks) that may communicate and interact with each other. The tasks are often precedence-related. Data files generated by one task are needed to start another task creating precedence constraints between them. Although this is the most common situation, the precedence constraints may also come from branching commands or loops. Such "complex" applications are called *workflow applications*.

Because of large amounts of computations and data involved, such workflows require high computing power to be executed efficiently. This computing power and storage capacity is now provided by grids. However, resource brokers have to cope with several applications submitted at the same time. These applications compete for the same resources and some scheduling must be done between them to ensure their correct execution and to avoid starvation while ensuring fairness.

In this paper, we propose to study the behavior of five different dynamic heuristics based on the key ideas of the well known heuristic HEFT [35]. We focus on dynamic workload where multiple new DAGs are submitted over time. These heuristics have been designed to improve the slowdown of the different applications sent from multiple users to the grid. We also observe the fairness of these heuristics which can be dramatically unfair in some case.

We have implemented these heuristics within the DIET middleware, a Network Enabled Server environment (NES) implementing the GridRPC API from the Open Grid Forum.

A new component called the MA_{DAG} has been added. The MA_{DAG} orchestrates workflows execution between users. The proposed architecture is scalable, robust, and is based on several DIET middleware features such as plug-ins scheduler and data persistency management. All the heuristics are validated and compared on the Grid'5000 research grid using target applications coming from bioinformatics.

The rest of the paper is organized as follows. Problem description and related work are discussed in Section 3. Then, in Section 4, we propose several heuristics for dynamic multi-workflow submission. Also, Section 5 describes the workflow manager called MA_{DAG} developed in the framework of the DIET grid middleware. Finally, before concluding and expose future works, Section 6 exhibits and tests heuristics behaviors on case study applications in a given scenario.

3 Problem Statement and Related Work

There are several ways to represent workflow [14], we mainly can define five types of representation: (a) *formal workflow* model with Petri nets and π-calculus. (b) *functional workflow* which separate function from data instance of the application. Workflow langages such as Swift, AGWL or GSFL falls into this category. (c) *services workflow* which are based on web service orchestration (Scufl, BPEL, ...) (d) *task graph* (i.e Directed Acyclic Graph noted DAG) defines all data dependencies between each tasks and avoid cycle (YML, DAGman, ...).(e) *executable workflow* which is a DAG where resource allocation has been established (Concrete Pegasus, XWFL, ...).

In this paper, we are interested only in the Task graph model (DAG) where data and computing time of each tasks are known. DAGs are described by a XML file which describes tasks and data dependencies. The DAG is a generic model of a scientific workflow

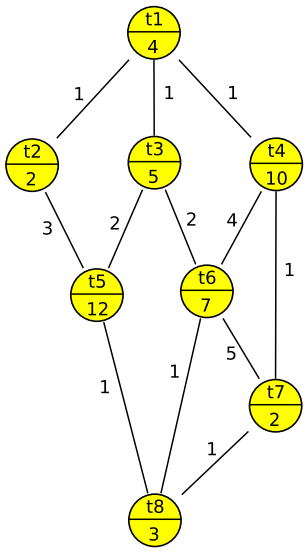


Figure 1: DAG example

application consisting of a set of tasks (nodes) among which precedence constraints exist. Formally, It is represented by a graph $G = (V, E)$, as illustrated in Figure 1, where V is the set of $|V|$ vertex (node or task) that can be executed on the set of the available

heterogeneous processors. E is the set of $|E|$ directed edges between the node that maintain a partial order among them. The partial order introduces precedence constraints, *i.e.* if edge $c_{i,j} \in L$, then task n_j cannot start its execution before n_i completes. Then the matrix D of size $|V| \times |V|$ denotes the communication data size, where $d_{i,j}$ is the amount of data to be transferred from n_i to n_j. The weight w_i of a node n_i represents its computation cost. The weight of an edge stands for the communication requirement between the connected tasks (the amount of data that must be communicated between them). In a given task graph, a root node is called an entry task and a leaf node is called an exit task. It is assumed that the task graph is a single-entry and single-exit one. If there are more than one exit or entry task, we can always connect them to a zero-cost pseudo exit or entry task with zero-cost edges. This will not affect the model.

Several surveys are already available about scheduling for distributed systems. Among them the following reports can be cited: workflows scheduling [38], workflow management systems [37], and scheduling algorithms for Grid Computing [10]. The DAG scheduling problem is *NP-complete* [13], but a huge amount of heuristics have been proposed. The heuristics are classified in four main categories [24]: clustering heuristics, Duplication based heuristics, meta heuristics, and list scheduling heuristics.

Clustering heuristics. Task clustering heuristic algorithms perform a sequence of clustering steps. Initially a task is assumed to be in a cluster then each step performs a refinement of the previous clustering so that the final clustering satisfies or is "near" to the original goals. The algorithms are non-backtracking, i.e., once the clusters are merged in a refinement step, they cannot be unmerged afterwards. A typical refinement step is to merge two clusters and *zero* the edge that connect them (Edge Zeroing algorithm [27]). Other techniques such as Dominant Sequence Clustering(DSC) [31] and Linear Clustering Method [22] exists.

Duplication based. Task duplication means scheduling a parallel program by redundantly allocating some of its tasks. The main idea is to reduce the start times of waiting tasks which can eventually improve the overall execution time of the whole program. Duplication based scheduling can be particularly useful for systems with high communication overhead and when there exists much more resources compared to the number of tasks. We can cite as illustration of this technics ; Task duplication-based scheduling Algorithm for Network of Heterogeneous systems (TANH) [1], Duplication Scheduling Heuristic (DSH) algorithm [23] and The Bottom-Up Top-Down Duplication Heuristic (BTDH) [6].

Metaheuristics. For an extensive survey on meta heuristic strategies see [21, 26]. Three strategies have been most popular and successful from among the meta heuristics over the years: simulated annealing, tabu search, and genetic algorithms. All this kind of heuristic try to find a good scheduling and allocation for Tasks graph by exploring the space of possible solutions. These techniques provide good solution but it take time to explore and generate the solution and it is not adaptable to dynamic environment such as grid systems.

List scheduling. The last method is List scheduling heuristics which carry out the following steps: (1) *task prioritizing* phase which gives an order of execution for tasks within the DAG. (2) *resources selection* phase which selects the suitable resource that optimize

a cost function. List scheduling heuristics are widely used algorithms. In fact, in the case of homogeneous processor without cost communication any list scheduling is within 50% of the optimum, thanks to Graham [16]. However, even if clustering or duplication heuristics are employed, then the scheduling and matching phases are comparable to list algorithm. There are many heuristics based on prioritization idea such as *Critical-Path-on-a-Processor* [34], *Generalized Dynamic Level* [29], *Iso-Level Heterogeneous Allocation* [2], *Levelized Min-Time* (LMT) [18], SDC [28],*etc.*

One of the well known heuristic is *Heterogeneous Earliest Finish Time* (HEFT) [35]. In this algorithm, tasks are ranked according to the Critical Path defined as a set of nodes and edges, forming a path from an entry node to an exit node, of which the sum of computation cost and communication cost is the maximum. The formula to obtain the rank of a task n_i is the following :

$$rank_u(n_i) = \overline{w_i} + max_{n_j \in succ(n_i)}(\overline{c_{i,j}} + rank_u(n_j)$$

where $\overline{w_i}$ is the mean time overall resources of the job n_i and $\overline{c_{i,j}}$ is the mean communication cost of the edge between n_i and n_j over all connexion of the resources. The set $succ(n_i)$ is the immediate successors of node n_i. Usually it is named as upward rank, b-level or simply critical path. This value is recursively computed from the exit node n_{exit}. The list of tasks is sorted according to non increasing of the upward rank. Then the heuristic selects the resource which minimizes the finish time of the task considering the data transfers of all parent tasks. The heterogeneity is tackled by considering the average cost between resources during tasks prioritization.

3.1 Related Work

Iverson et al. [19] present a multiple workflow applications framework based on a hierarchical matching and scheduling architecture. They adopt a decentralized scheduling strategy where each application makes its own scheduling decision during the allocated time slots. Different scheduling time policies are compared for their impacts on overall resource utilization, but they do not take into account slowdown measurement which is more a user oriented metric. Furthermore, their study analyses this problem with simulation and there are no implemented version of their architecture.

Zhao et al. [41] also propose composition-based approaches in order to merge multiple DAGs into a single DAG before applying an algorithm designed for fairness. They consider several static algorithms where all DAGs are known in advance.

Similarly Hönig et al. [17] describe a meta-scheduler for multiple DAGs, which suggests to merge multiple DAGs into one to improve the overall parallelism and optimize idle time of resources. However, these efforts are limited to the static case and they do not deal with dynamic workloads. They also do not propose implemented version of their work.

Duan et al [11] have published a scheduling algorithm based on the adoption of game theory and the further idea of sequential cooperative game. They provide two novel algorithms in order to schedule multiple DAGs. Their algorithms work properly for applications which can be formulated as a typical and solvable game.

One last recent work [39] tackles problem of dynamic scheduling multiple DAGs from different users. They expose a similar approach from Zhao et al. [41] without merging DAGs. Their algorithm is similar to our *G-heft* algorithm and their simulation results obtained with generated dummy graphs confirm our results obtained on real case applications.

On the other hand, there exist many workflow management systems such as Taverna [33], Triana[25], Pegasus/DAGMan [30, 32], GridFlow [4], ASKALON [12], MOTEUR [14, 15], *etc*, which implement different heuristics (see the taxonomy of workflow system written by Buyya et al [37]). Most of them implement list scheduling heuristics with eventually some clustering algorithms [30] or when the workflow is expressed as a *functional workflow* [33, 15], it provides a scheduling called just-in-time scheduling [8], where a mapping phase selected a free and suitable resource to execute a task when it just becomes ready. To the best of our knowledge, the workflow enactors deal with multiple DAGs submission by making several call to the scheduler, but there do no exist specific strategy to take into account several users at the same time.

4 Online Multi-DAGs Scheduling Heuristics

We have exposed the main technics used for scheduling a DAG and more precisely using the HEFT heuristic. Now we will extend this to the online case with several DAGs are submitted over time. The problem is the following. There are several users who want to submit several jobs (DAGs) with different data and parameters to a set of heterogeneous grid resources. Jobs are instances of applications, and applications are sets of tasks with data or time dependencies (DAG). Let's extend the definition of the problem with several applications which are submitted following its own arrival time function.

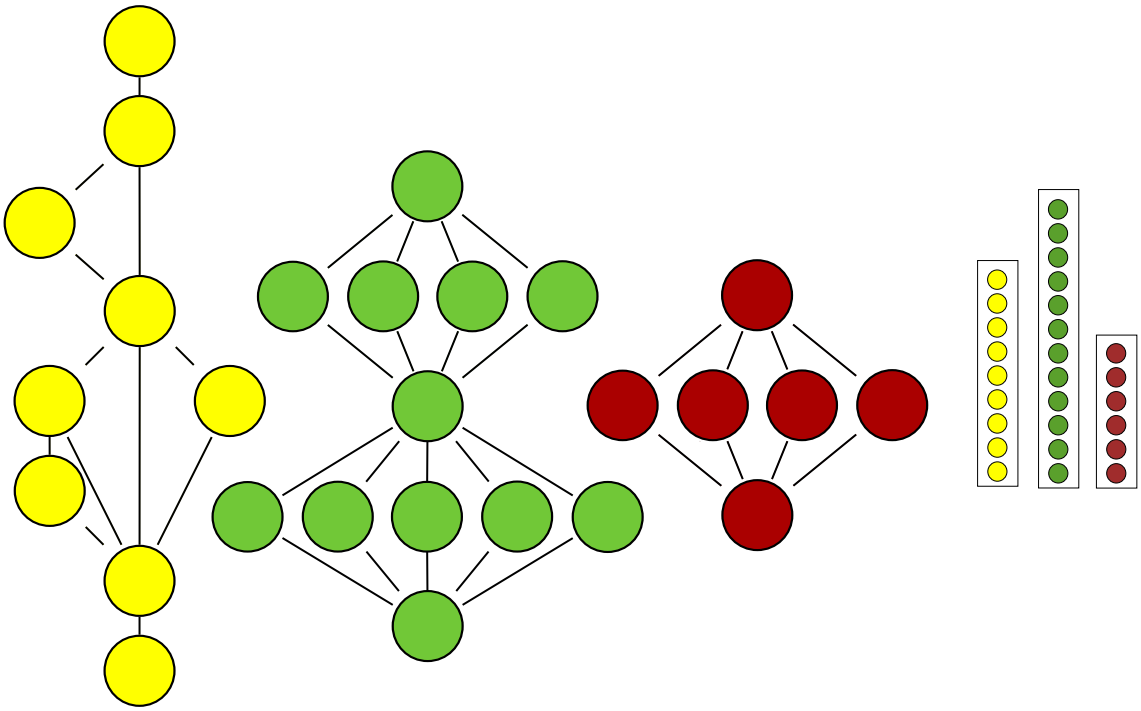


Figure 2: Multiple workfow applications submission

* $(a_1, a_2, a_3, ...)$ the set of applications.
* $(r^{a_1}, r^{a_2}, r^{a_3}...)$ the arrival time of each applications.
* $G^{a_n} = (V^{a_n}, E^{a_n})$ is the directed acyclic graph of application a_n.
* V^{a_n}: the tasks of the application a_n, $(v_i^{a_n})_{i \in [,|V^{a_n}|]}$.

* E^{a_n}: the dependencies between tasks $v_i^{a_n}$ and $v_j^{a_n}$, $(e_{i,j}^{a_n})_{(i,j)\in[1,|V^{a_n}|]}$.
* w^{a_n} is the cost of the task $v_i^{a_n}$.
* $d_{i,j}^{a_n}$ is the dependency cost between tasks $v_i^{a_n}$ and $v_j^{a_n}$

In the case of multi-DAGs, we introduced two different priorities. The first one is the intra-DAG priority which sets the order of task within a DAG, then the inter-DAG task priority which set the priority of a task relatively to the tasks of other applications (other DAGs). All the tasks of the same DAG have the same intra-DAG task's priority value. We have decided to use the priority of the HEFT heuristic for the intra-DAG priority, because it gives good results and there exists several paper which mentioned its good behavior.

$$\forall a_n \in \mathcal{A}, rank_{intraDAG}(t_i^{a_n}) = rank_u(t_i^{a_n})$$

$$rank_u(t_i^{a_n}) = \overline{w_i}^{a_n} + \max_{t_k^{a_n} \in succ(t_i^{a_n})} (\overline{c_{i,k}}^{a_n} + rank_u(t_k^{a_n})$$

Similarly, the mapping phase stays unchanged. The scheduler selects the resource which minimizes the finish time of the task considered. Then we explore several ways to set the inter-DAG priority. All of the following heuristics respect the two ideas of the list-scheduling, give a priority to tasks into a DAG, then select a resource that optimize a cost function for the highest priority task unscheduled.

Algorithm 1 Online list scheduling algorithm

1: Each time a DAG d_s is submitted :
2: - compute ranks (inter-DAG and intra-DAG) of each task of the DAG d_s.
3: - Sort the list U of tasks by decreasing order of their rank.
4: **while** there are unscheduled tasks **do**
5: - select the first unscheduled task t from the list U.
6: - choose the suitable server s for task t.
7: - allocate t on s.
8: **end while**

basic. This heuristic is called BASIC because it does not take into account that fact that several DAGs could be submitted over time. The inter-DAG priority is thus not set. There is a separate management of each DAG and the HEFT [35] heuristic is applied. However, the resources are shared.

G-heft: Global Heterogeneous Earliest-Finish-Time. The inter-DAG priority is equal to the intra-DAG priority. This is like each time a new DAG is submitted to the system, a meta-DAG of all the DAGs currently executed by the system is constructed and then the HEFT heuristic is applied to this meta-DAG. It has been presented in [41] for the static case. In fact, the construction of the meta-DAG is not necessary, and only the tasks of the new submitted DAG need to be ranked and the ranked tasks must be inserted into the sorted list of all tasks in the scheduler.

$$\forall a_n \in \mathcal{A}, rank_{interDAG}^{G-HEFT}(t_i^{a_n}) = rank_{intraDAG}(t_i^{a_n}) = rank_u(t_i^{a_n})$$

The problem with this heuristic is that one can have DAGs which will never end if the submission of DAGs is constant. This scheduling heuristic induces a starvation for low

ranking tasks, (i.e the tasks at the bottom of the DAG). Because of the ranking function of the tasks, the rank of one task will remind constant along the time. And so the low rank tasks should never be scheduled if there is a new DAG which is submitted with tasks with higher rank. This is why we have introduced a weight on the task based on the oldness of their submission.

aging G-heft. This heuristic (Aging Global Heterogeneous Earliest-Finish-Time) follows exactly the same scheme of G-HEFT, but a coefficient is introduced into the inter-DAG priority computation which depends on the oldness of the DAG submission. This method can potentially avoids the starvation phenomena for low ranking tasks.

$$\forall a_n \in \mathcal{A}, rank^{\text{Aging G-HEFT}}_{interDAG}(t_i^{a_n}) = rank_{intraDAG}(t_i^{a_n}) \times f(\text{age})$$

Two types of coefficients have been explored:

inter-DAG priority depends on the oldness of the application divided by the estimated makespan of the application. This weighting enables to take into account oldness and length of the application.

$$f(\text{age}) = 1 + \frac{\text{age}}{makespan}$$

The exponential value of the previous factor is used.

$$f(\text{age}) = e^{1+\frac{\text{age}}{makespan}}$$

fcfs heuristic. First Come, First Served and Shortest Remaining Processing Time heuristic change from the previous heuristics only because they take into account the oldness of the application.

$$\forall a_n \in \mathcal{A}, rank^{FCFS}_{interDAG}(t_i^{a_n}) = r_i^{a_n}$$

srpt heuristic. Shortest Remaining Processing Time heuristic set the inter-DAG priority with the sum of the estimated processing time remaining with the unfinished tasks of the DAG.

$$\forall a_n \in \mathcal{A}, rank^{SRPT}_{interDAG}(t_i^{a_n}) = 1/ \sum_{t_i \text{unfinished}} \overline{w_i}^{a_n}$$

The idea behind this heuristic is to finish as soon as possible DAGs which have been submitted by selecting the tasks that belong to the lowest time consuming application.

foft heuristic. The key idea of Fairness On Finish Time (FOFT) is to sort the list of applications (i.e. the DAGs) according to their slowdown value. Let's define the slowdown or stretch of an application as the ratio of its flow time to its processing time.

$$slowdown^{a_n} = \frac{C^{a_n} - r^{a_n}}{Cp^{a_n}}$$

$$\forall a_n \in \mathcal{A}, rank^{FOFT}_{interDAG}(t_i^{a_n}) = slowdown^{a_n}$$

Processing time (Cp^{a_n}) represents the service time of the application a_n. This is the critical path of first unexecuted task in the DAG. The *flow time* ($C^{a_n} - r^{a_n}$) is the difference

between the arrival time r^{a_n} and the completion time C^{a_n} of this application a_n. In order to compute the completion time, we assume that the completion time of one application is the estimated time at the current time if we choose to fully schedule this application. This heuristic selects the task which belong to the DAG has have been lagged the most.

To summarize we have six different heuristics based on the key ideas of the list scheduling HEFT, all of them are non-clairvoyant (*e.g* on-line). Five of them are multi-workflow oriented and explore some ways of setting an inter-DAG priority onto tasks.

5 Implementation

In order to test all previous heuristics we have implemented them into The Distributed Interactive Engineering Toolbox (DIET) [5, 9] middleware environment.

This project is focused on the development of a scalable middleware with initial efforts focused on distributing the scheduling problem across multiple agents. DIET consists of a set of elements that can be used together to build applications using the GridRPC paradigm. This middleware is able to find an appropriate server according to the information given in the client's request (*e.g.*, problem to be solved, size of the data involved), the performance of the target platform (*e.g.* server load, available memory, communication performance), and the local availability of data stored during previous computations. The scheduler is distributed using several collaborating hierarchies connected either statically or dynamically (in a peer-to-peer fashion). Data management is provided to allow persistent data to stay within the system for future re-use. This feature avoids unnecessary communication when dependencies exist between different requests (*e.g.* in case of same or different requests using same data will be executed on the same server). Servers have the possibility to launch several tasks either in a time-shared manner, either sequentially, making servers buffer some work, with a parameter we can defined number of concurrent jobs at a given moment on a given server [7].

An overview of the DIET architecture is shown in Figure 3. This shows the main components of the DIET hierarchy: the Master Agent (MA), the Local Agents (LAs),the Server Daemons (SeDs), and the MA_{DAG} which is responsible for workflow management and scheduling. The MA_{DAG} can be considered as outside of the platform, there can be several MA_{DAG} connected to the platform through the MA. This agent is not needed by other components except for the client, but the client can also manage the execution of the workflow by itself. It is important to notice that even if the MA_{DAG} controls the execution of the workflow, there is no input/output data which is sent from client to MA_{DAG} but only data handles.

Workflow submission. A workflow application is submitted to the MA_{DAG} as an XML file containing the description of all tasks with their input/output ports and the connections between these ports that represent the data flows.

The MA_{DAG} agent runs simultaneously a multi-workflow scheduler process that is unique and one HEFT scheduler process for each new dag submission. This HEFT scheduler implements the intra-workflow scheduling heuristic which is HEFT in our case. Therefore the step assigned to this process is first to parse the XML file and analyze the DAG structure, then to get estimations of task durations from the SeDs thanks to plug-in scheduler and apply the HEFT heuristic to order the tasks of the DAG. This intra-workflow scheduling phase defines an intra-workflow priority for each task (the "HEFT Priority").

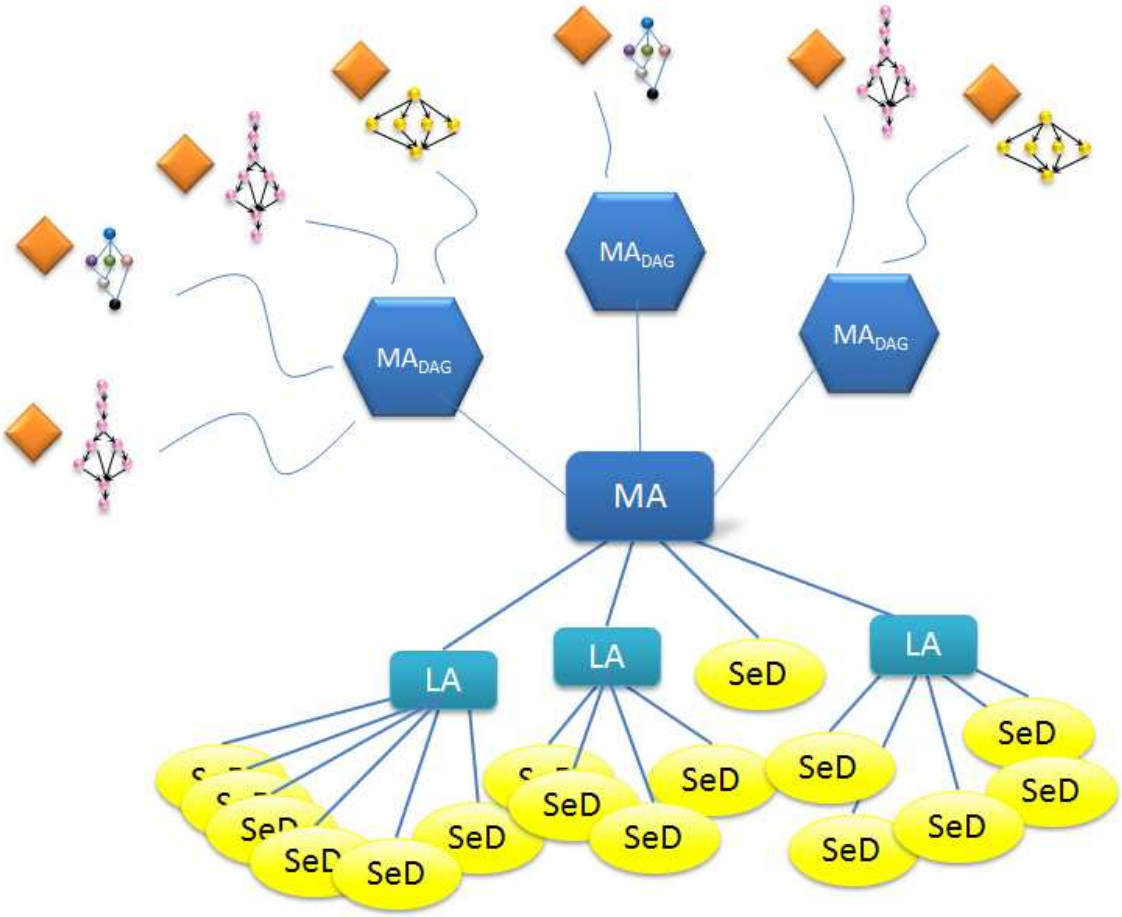


Figure 3: Multi-MA$_{\mathrm{DAG}}$ architecture.

At this stage the tasks are only ordered but not mapped to a specific resource on the grid because this mapping depends on the next phase. The output of the first phase for a given dag is an ordered list of tasks that are ready to be executed. This list will be continuously updated each time a task terminates in order to include the new ready tasks which are inserted in the list according to their HEFT priority. The multi-workflow scheduler takes all the tasks lists corresponding to the dags being executed and can apply either a task-based policy or a dag-based policy to define the order of execution for the tasks.

If we choose the task-based policy then all ready tasks of all dags will be taken into account when we apply the inter-dag prioritization heuristic, which means that several tasks of the same DAG may be selected before tasks of other DAGs. If we choose the DAG-based policy then only the top priority task inside each list will be selected for the inter-dag prioritization because the inter-dag priority will not depend on the task's HEFT priority but only on the DAG's current status.

According to the policy we therefore define a set of ready tasks from different DAGs, compute for each the inter-DAG priority and order them according to this priority. This defines an execution list for the current round. The scheduler takes the highest priority task in this list and tries to find the best resource in terms of earliest finish time for this task. This is done by sending a request for submission to the grid middleware. If a resource is found then the MA_{DAG} will send a message to the client that submitted the corresponding dag with the reference of the selected resource. The client will execute the task on this resource and send back a message to the MA_{DAG} when the execution is finished. The scheduler tries to assign resource to as many tasks as possible in the execution list, so the round terminates if either no more ready tasks are available in the execution list or no more resources are available.

The scheduler goes into "sleep" mode until a new round is started when it receives a message of task termination or a new DAG submission.

6 Experiments

We have tested the behavior of the implemented scheduling heuristic into the MA_{DAG} of the DIET middleware with three applications coming from bioinformatics. Those applications named *pipeAlign*, *Gee*, *Maxdo* are described briefly in the following section.

6.1 Multiple Alignment : *PipeAlign*

Homology modeling represents a powerful starting point for studies of the relationships between a sequence particularly when based on Multiple Alignment of Complete Sequences (MACS). The *PipeAlign* application is a succession of steps which produce a high quality protein alignment starting from the query sequence. First Ballast (homology search) runs BlastP to search for homologies in a proteins database (UniProt, SwissProt, Uniref90,...). Then BlastP search is performed using the query sequence within the PDB database. The PDB with the best score associated with is defined as the closest PDB. DbClustal (sequences alignment) build the multiple alignment of the sequence detected (MACS). Rascal (alignment refinement) scans the MACS to identify misaligned residues and blocks. NorMD (objective function) evaluate the quality of both DbClustal and Rascal alignments. Leon (unrelated sequence removal) process the highest scored MACS and removes potential weakly related or highly fragmented sequences to generate the final MACS, which is also scored by NorMD. Finally, DPC/Secator (clustering step) clusters subfamilies.

Figure 5 shows the workflow defined by the *PipeAlign* application. This application takes as input a protein sequence and user parameters, then computes all steps of the worklfow. The precedences constrains symbolized by the edge between tasks represent data dependency: output files are required for the execution of the next step. We assume that the data bases (i.e. Uniref90, Uniprot, SwissProt, ...) are already deployed on the execution server, and we do not take into account the movement of this read only database.

6.2 Proteins docking application : *MAXDo*

This application is used to detect protein-protein and protein-DNA interactions. Identifying pairs or larger complexes of functionally interacting proteins, or determining the binding of a protein to a DNA sequence or to a ligand are fundamental problems in biology with immediate consequences in drug design. *MAXDo* is a docking program which computes optimal interaction geometries using multiple energy minimizations with a regular array of starting positions and orientations.

Figure 5 represents the workflow of the docking application. The first task of the diamond workflow takes the two proteins to dock and computes the parameters info needed to determine the search space of binding sites. Then it divides the search space into 4 tasks and sends the proteins files and the parameters to the 4 docking tasks. After all the docking tasks are finished the last task collects the result files of the docking computation and computes statistics information about the minimization.

6.3 Gene Expression Evolution application : *GEE*

This application annotates human genes according to not only their expression in neurological or muscular tissues, but also the expression of their homologs in other species. This method allows to propose new targets for furthers investigation, by annotating the human genome with the respect to implication in normal or pathological neuromuscular processes. The scientific aims are to build an efficient workflow for annotation based on comparative analysis in gene expression, and to highlight the conservation or evolution of gene expression in animals, for a well defined biological process (neuromuscular development and function).

The methodology used consists of the following steps: mapping of expression data to animal genes by sequences similarity; integration of the expression information into a multi-species ontology; clustering of genes in gene families, for which alignment and phylogenies are build; guided automatic selection of gene families with relevant expression patterns. Annotating the human genome for neuromuscular processes, and especially the highlighting of gene families involved in abnormal processes (diseases, mutation, treatments ...), provide promising targets for further biomedical investigation. This is notably interesting for discovery of genes involved in rare genetic diseases, not easily amenable to classical approaches.

Figure 5 represents the workflow of the Gene Expression Evolution application.

6.4 Settings and scenario

In order to validate the behavior of the heuristics presented, we have set a framework that reproduce the behavior of a light grid environment where several user shared resources. Figure 4 illustrates the deployed architecture. All services needed by the three applications are declared in the SeDs which are all executed on different resources. The $\mathrm{MA}_{\mathrm{DAG}}$ and

	MAXDo	PipeAlign	GEE
makespan	33 s	1 min 36 s	2 min 3 s
$\sum w$	1 min 20 s	2 min 5 s	3 min 20 s

Table 1: Makespans for each application executed alone with 4 resources (mean over 3 runs).

the MasterAgent are deployed on the same host and one server per SeD which runs tasks one by one. If there are more than one task assigned to the same SeD, this SeD puts the additional tasks into queue, then the task is selected with the FIFO policy (First In, First Out). For all the 14 services; the 3 services of *MAXDo* application, the 7 services of *PipeAlign* and the 4 services of *GEE*, the SeD can estimate the time needed to execute the task based on its performance and task requirement.

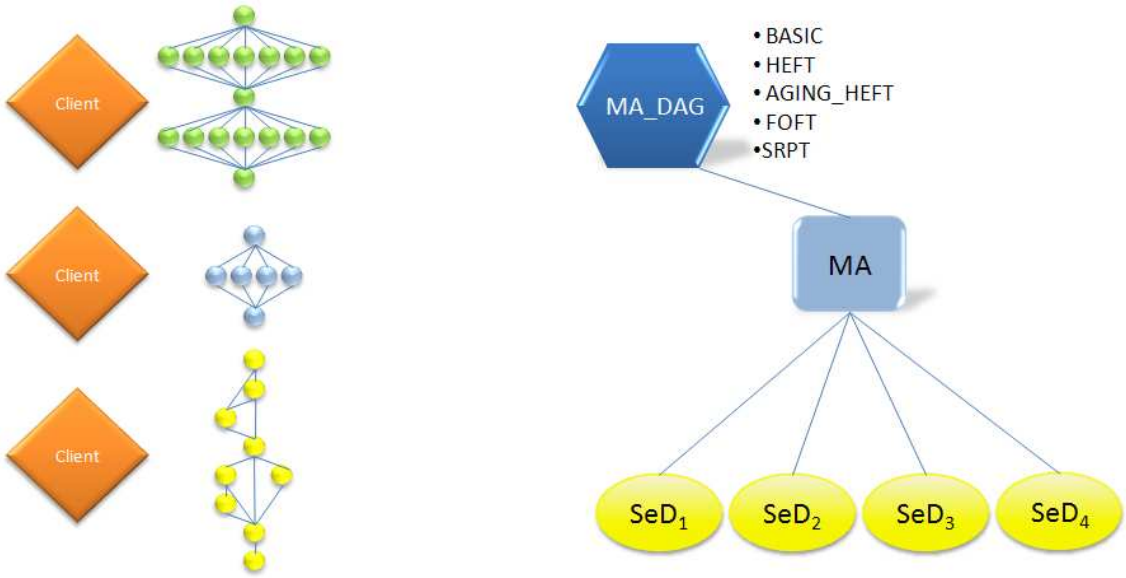


Figure 4: Evaluation platform

One may have noticed that the test environment is quite small with only 4 servers able to compute services, but all the multi-workflow heuristics have sense only when user's jobs are competing for having resources. If the number of resources is greater than the number of concurrent tasks execution, all the heuristics have the same behavior and schedule DAGs just like the HEFT heuristic. That's why we use this small environment in order to exhibit the behavior of the heuristics in a shared environment. This is not a limitation of the DIET middleware that has already ran experiment on over more than 1000 processors to execute 45000 clients' requests.

The mean execution time and the valuation of the HEFT ranking function for each task of the application is given in Figure 5. Furthermore the mean makespan of each application executed alone is given in Table 1.

The experiment follows the following scenario:

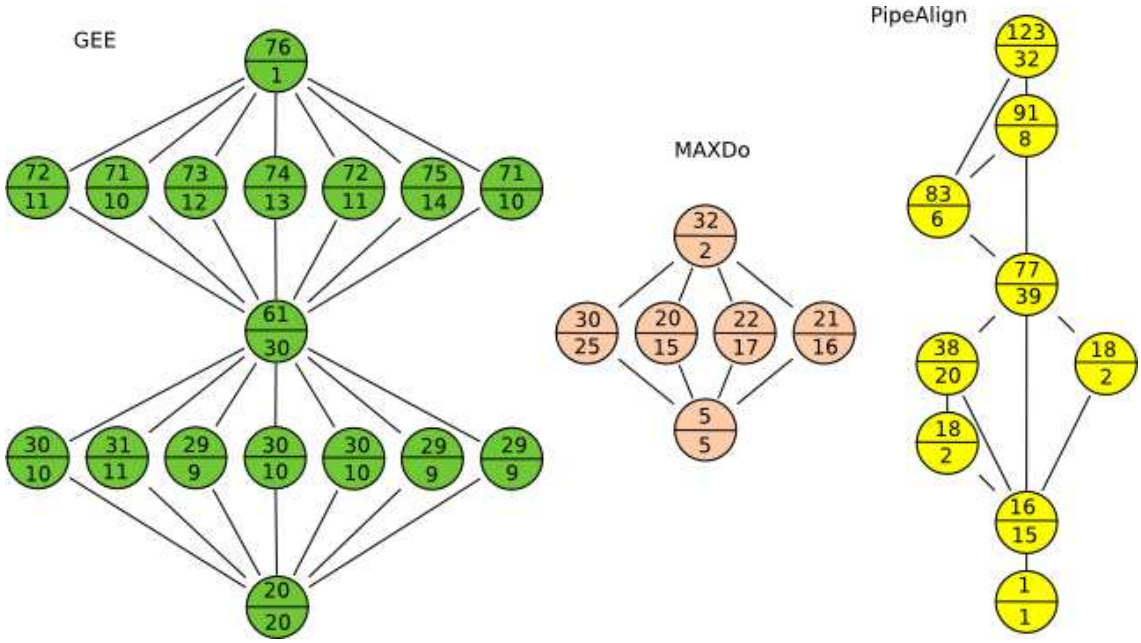


Figure 5: Valuated graph for *GEE*, *MAXDo*, and *PipeAlgin* applications.

- At r_0, start of the experiment, the first client submits its workflow application. Then, every 12 seconds a new client submits its DAG. The type of application is selected randomly among the applications that have not been submitted. The last submission of the first series happens at r_0 + 600s. The first series make 10 requests of the *GEE* application, 25 of *MAXDo* and 16 of *PipeAlign*;
- At r_0+1200s, an other random series of submission is done with the same number of DAGs of each type. The last submission is done at r_0+ 1800s.

The parameters used for this experiment were selected in order to have the same amount of processing time asked for each applications type:

$$\sum w^{GEE} = \sum w^{MAXDo} = \sum w^{PipeAlign}$$

Figure 6 shows the number of DAGs managed by the MA_{DAG} from the begin to the end of the experiment for each heuristic, and Figure 7 exhibits the number of tasks ready to be executed. The red cross stands for the submission date r^{a_n} and the hight of the red cross represents the number of tasks of the submitted applications (*e.g.* 17 for *GEE*, 6 for *MAXDo*, 9 for *PipeAlign*).

The Figures 6 and 7 express several behaviors :

- For all heuristics, the makespan of the experiment is similar. In particular, if this time is compared to one lower bound of the makespan (considering 4 processors and all tasks of application as divisible tasks without any constraint), we observe that all heuristics are between 2% to 3.5% near this theoretical lower bound;
- Those heuristics which set a inter-DAG priority (FOFT, FCFS, SRPT) and AGING e G-HEFT, end DAG earlier;
- Conversely, the G-HEFT, AGING G-HEFT achieve the complete DAG execution almost at the same time at the end of the experiment.

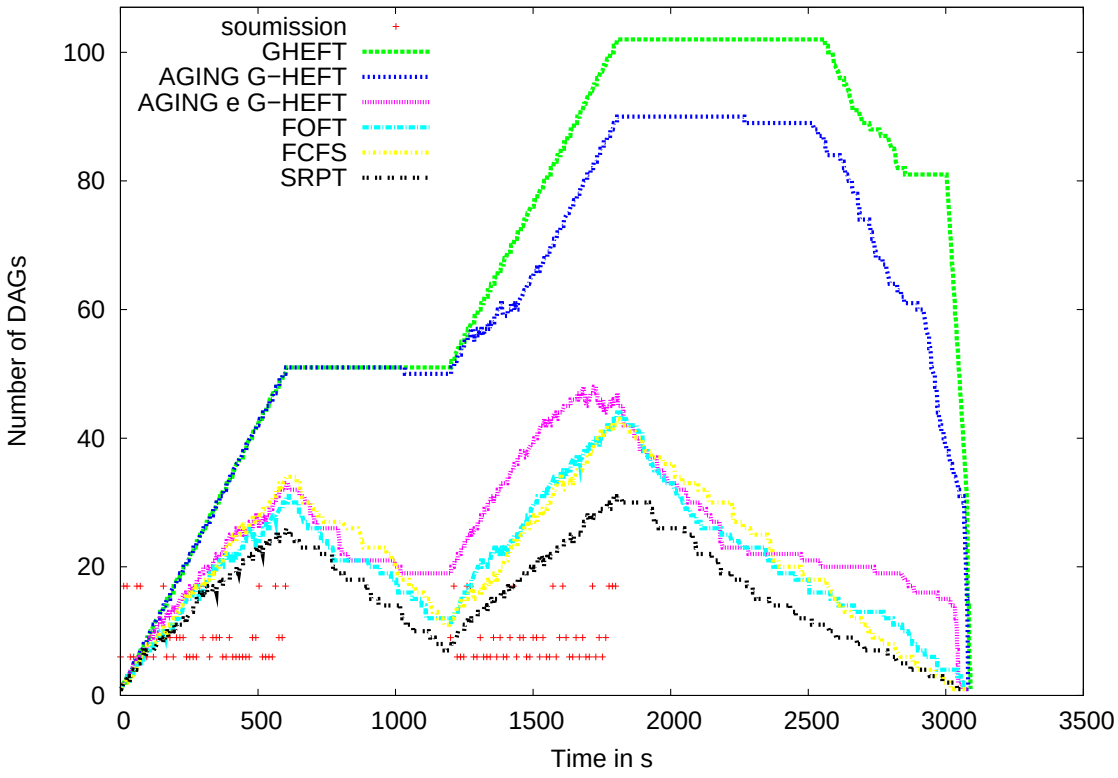


Figure 6: Number of DAGs submitted along the experiment.

- Introducing the oldness into the inter-DAG priority improves the number of DAGs ending earlier increases (G-HEFT compared to AGING G-HEFT);
- Figure 7 illustrates the wave progression of the heuristic G-HEFT. Indeed this scheduling policy gives the inter-DAG priority equal to the intra-DAG priority, As in this experiment, all the submitted task graphs of one application are similar, it produces a scheduling that order all the "same" tasks with the same priority and so a wave progression of task execution. Introducing the oldness changes this behavior a bit.

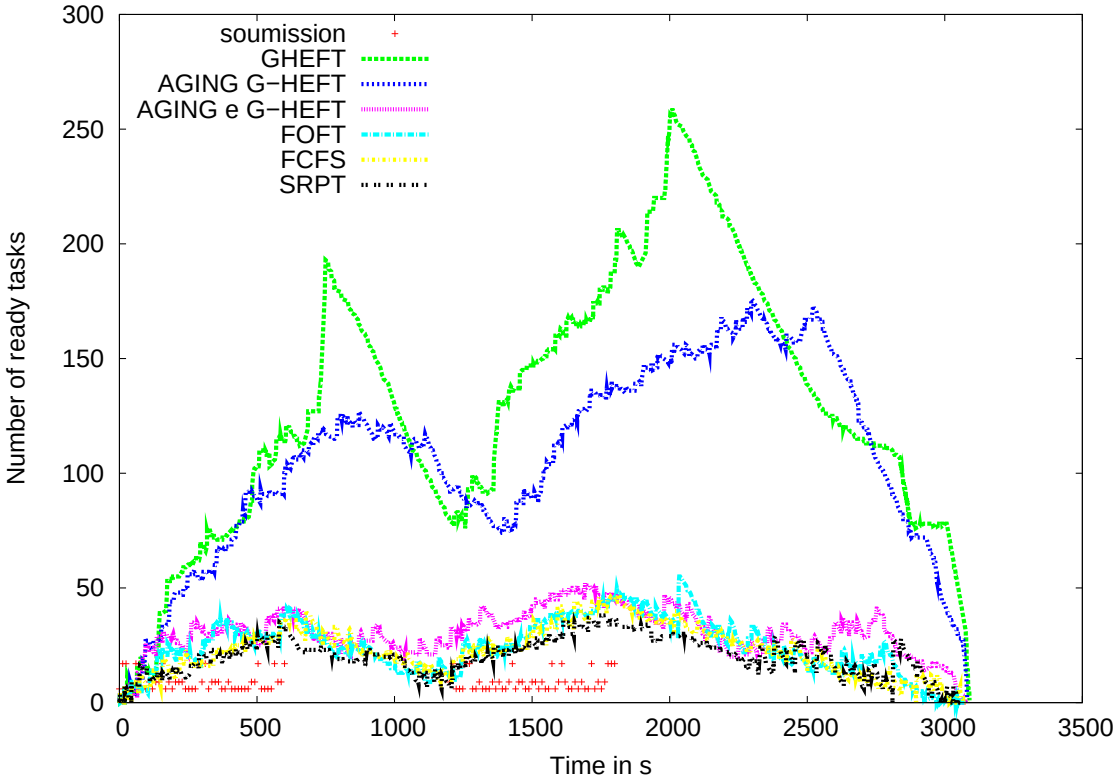


Figure 7: Number of tasks ready to execute.

Figures 8 and 9 present the box & whiskers plot [36] of slowdown statistics. The length of the box stands for the interquartile range ($Q_3 - Q_1$), the stick and point into the box stand respectively for the median and the mean value, the two whiskers stand for the minimum and the maximum. As for the FOFT heuristic, the slowdown is defined as the ratio between the makespan of the application in the experiment and the estimated makespan if executed alone (see Tab 1). More the value is closer to 1, less the application has been slow.

Without any surprise, the G-HEFT heuristic has the worst slowdown (highest one). Namely this heuristic could never end any DAG execution if there was a continuous DAGs submission. We also observe the fact that introducing oldness into the inter-DAG priority decreases the slowdown values. Furthermore, in the selected scenario, the SRPT heuristic gets the lowest mean slowdown followed by FOFT, AGING e G-HEFT, and FCFS. Indeed, the selection criterion of SRPT promotes the *MAXDo* application which needs less comput-

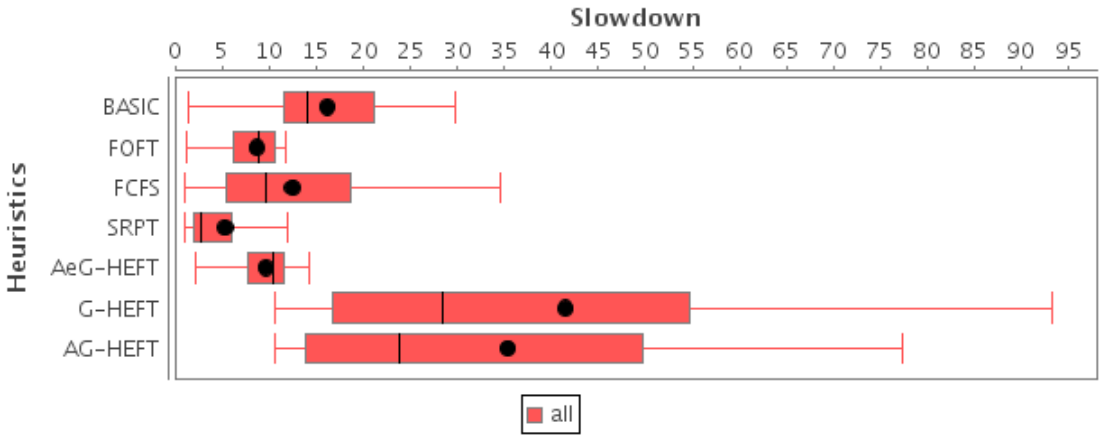


Figure 8: Box & Whiskers plots of slowdown for all application types without distinction.

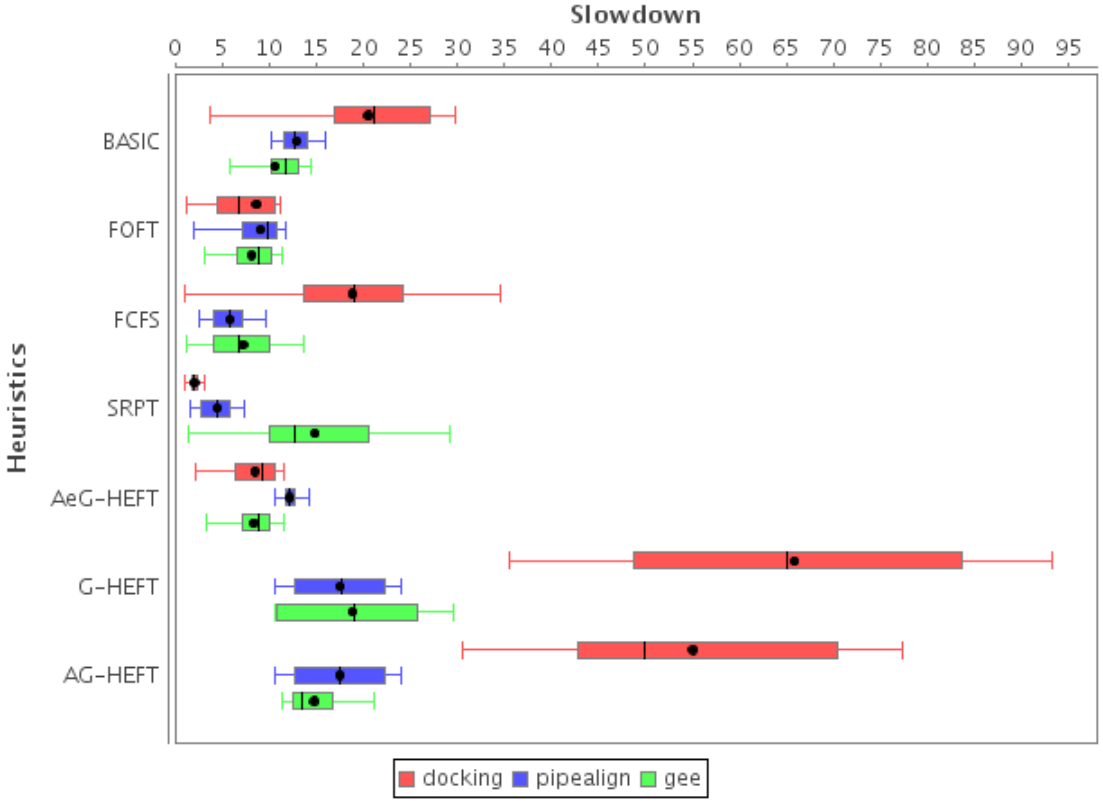


Figure 9: Box & Whiskers plots of slowdown for each application (*MAXDo*, *pipeAlign*, *GEE*);

ing time. In the same way, the *MAXDo* application has also the smallest makespan (see Table 1) and consequently it is much more slowdown sensitive. In particular, Figure 9 illustrates the homogenization tendency of slowdown for the FOFT heuristic contrary to SRPT which favor short application and FCFS which unfavor *MAXDo* compare to other. Finally, the AGING e G-HEFT heuristic aggregates advantages of FCFS, FOFT and SRPT by setting the inter-DAG priority which highly (exponential) depends on the ratio age/makespan of the application (see also Figure 6). Indeed, the application with a small makespan have the ratio age/makespan much more sensitive compare to other. In addition, the exponential function depending on age gives a behavior similar to the FCFS.

Now, let's take care of the fairness. There are several ways to define fairness [3, 20, 40]. This value considers the difference between application regarding some metric. Here we define the fairness as the dispersion around the slowdown suffers by each application. Several values give measurement of dispersion and so fairness.

- range: the length of the smallest interval which contains all the data;
- interquartile range: the difference between the third and first quartiles;
- average deviation: arithmetic mean of the absolute difference compare with the mean;
- standard deviation: quadratic mean of the difference compared with the mean.

The box & wishkers plot of Figures 8 and 9 shows most of the precedent dispersion values. We observe that AGING e G-HEFT, FOFT have the smallest dispersion value which exhibits the fairness of these heuristics. Moreover, they have a better mean slowdown compared with BASIC heuristic.

In brief, with this experiment scenario, it appears that the experiment makespan stays comparable for each heuristics and it is near one theoretical lower bound (between 2% and 3,5%). Furthermore, the best heuristic regarding the mean slowdown is the SRPT. For this scenario, it can be explained because of the high number of small applications (*e.g. MAXDo*). Also it is better to promote small applications in order to get a low slowdown, insofar as those small applications are much more sensitive to slowdown. Concerning fairness, AGING e G-HEFT and FOFT heuristics get the smallest dispersion of slowdown values and it keeps a mean slowdown near the best one obtains with SRPT.

6.5 Others experiments and scalability

We have also launched many other experiments in order to qualify the behavior of our heuristics. For example, we have tested different order of arrival time, different application numbers, different sizes of the applications parameters (processing time), and different resources. The conclusion of the commented scenario appears to be quite general and valid in most cases, but of course, there are counterexamples and bad situations for some heuristics.

The MA_{DAG} implementation is quite robust and it can handle many workflows execution. But as it is a central point of connection for clients that share resources, it can become a bottleneck. As presented in Figure 3, there can be several MA_{DAG} connected to the DIET hierarchy. In this case, the multi-workflow heuristics would be inefficient unless all clients that submit jobs to each MA_{DAG} use the same resource exclusively. In any case, the implementation proposes a valid model to enable multi-workflow scheduling.

7 Conclusion and future works

This paper attacked the problem of scheduling dynamically several workflow applications sent in an online fashion to a grid environment. The application execution are concurrent and share the same resources. Five new heuristics named G-HEFT, AGING e G-HEFT, SRPT, FCFS, and FOFT have been proposed. All of them are based on the key ideas of the classical list scheduling heuristic HEFT. We have proposed to introduce the notion of inter-DAG and intra-DAG priorities. In general, it seems that G-HEFT should be avoid in the context of dynamic submission of workflows among time if we take care about slowdown of applications. Nevertheless it can produce good performance in term of experiment makespan. On the other hand, FOFT and AGING e G-HEFT heuristics are still the most fair heuristic and when the proportion of small applications is high, it can be appropriate to use the SRPT heuristics.

In the next step, we plan to explore other key ideas in the ordering and mapping phases. For example, we can use other ranking function to deal with heterogeneity such as the SDC [28] heuristic which takes into account the effect of Percentage of Capable Processors. We also can change the mapping function which minimizes the finish time of the task when it selects the resource. Moreover, we can deal with workflow applications that need multi-processors assignment. Finally, we have planned to extends the behavior of the $\mathrm{MA}_{\mathrm{DAG}}$ to enable functional workflow description where the whole DAG is not known in advance and test with other kind of applications where the workflow is dynamically build among the execution.

8 Acknowledgement

This work is partially funded by the GWENDIA project (http://gwendia.polytech.unice.fr, contract ANR-06-MDCA-009) from the French National Agency for scientific Research (ANR). We are grateful to the Décrypthon project set up by CNRS, AFM, and IBM, which have provided scientific applications that have been used to validate the heuristics and the implementation.

References

[1] Rashmi Bajaj and Dharma P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):107–118, 2004.

[2] O. Beaumont, V. Boudet, and Y. Robert. The iso-level scheduling heuristic for heterogeneous processors. In *PDP'2002, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*. IEEE Computer Society Press, 2002.

[3] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing Syst. Theory Appl.*, 53(1-2):65–84, 2006.

[4] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow: Workflow management for grid computing. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 198, Washington, DC, USA, 2003. IEEE Computer Society.

[5] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2006. To appear.

[6] Y.-C. Chung and S. Ranka. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 512–521, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[7] Holly Dail and Frédéric Desprez. Experiences with hierarchical request flow management for network-enabled server environments. *International Journal of High Performance Computing Applications*, 20(1), February 2006.

[8] Ewa Deelman, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 11–20, 2004.

[9] DIET. Distributed Interactive Engineering Toolbox. http://graal.ens-lyon.fr/DIET.

[10] Fangpeng Dong and Selim G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, Queen's University, School of Computing, January 2006.

[11] R. Duan, R. Prodan, and T. Fahringer. Performance and cost optimization for multiple large-scale grid workflow applications. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.

[12] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. Askalon: A grid application development and computing environment. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131, Washington, DC, USA, 2005. IEEE Computer Society.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[14] T. Glatard. *Description, Deployment and Optimization of Medical Image Analysis Workflows on Production Grids*. PhD thesis, Université de Nice Sophia-Antipolis, Sophia-Antipolis, November 2007.

[15] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and Efficient Workflow Deployement of Data-Intensive Applications on Grids with MOTEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*, 2008.

[16] R. L. Graham. Bounds on multiprocessing timing anomalies. *Siappmat*, 17:263–269, 1969.

[17] U Hönig and W. Schiffmann. A meta-algorithm for scheduling multiple dags in homogeneous system environments. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, 2006.

[18] M. A. Iverson, O. Gregory, and J. Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *4th Heterogeneous Computing Workshop (HCW '95)*, pages 93–100, 1995.

[19] M A Iverson and F Ozguner. Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment. *Distributed Systems Engineering*, 6(3):112–120, 1999.

[20] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *ArXiv Computer Science e-prints*, September 1998.

[21] James P. Kelly. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[22] S.J. Kim and J.C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. In *Int'l Conf. Parallel Processing*, volume 2, pages 1–8, 1988.

[23] Boontee Kruatrachue and Ted Lewis. Grain size determination for parallel processing. *IEEE Softw.*, 5(1):23–32, 1988.

[24] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[25] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana: A graphical web service composition and execution toolkit. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 514, Washington, DC, USA, 2004. IEEE Computer Society.

[26] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY., 1995.

[27] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.

[28] Z. Shi and J. J. Dongarra. Scheduling workflow applications on processors with different capabilities. *Future Gener. Comput. Syst.*, 22(6):665–675, 2006.

[29] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993.

[30] Gurmeet Singh, Mei-Hui Su, Karan Vahi, Ewa Deelman, Bruce Berriman, John Good, Daniel S. Katz, and Gaurang Mehta. Workflow task clustering for best effort systems with pegasus. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–8, New York, NY, USA, 2008. ACM.

[31] A. Gerasoulis T. Yang. Dsc: Scheduling parallel tasks on an unbounded number of processors. *EEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[32] Condor Team. "the directed acyclic graph manager". http://www.cs.wisc.edu/condor/dagman.

[33] The Taverna Team. Taverna workbench : a software tool for designing and executing workflows. http://taverna.sourceforge.net.

[34] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 3, Washington, DC, USA, 1999. IEEE Computer Society.

[35] H. Topcuouglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[36] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, New York, 1970.

[37] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[38] Jia Yu and Rajkumar Buyya. Workflow scheduling algorithms for grid computing. Technical Report GRIDS-TR-2007-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 2007.

[39] Zhifeng Yu and Weisong Shi. A planner-guided scheduling strategy for multiple workflow applications. *Parallel Processing Workshops, International Conference on*, 0:1–8, 2008.

[40] H. Zhao and R. Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *Proceedings of the 15th Heterogeneous Computing Workshop (HCW)*, Rhodes Island, Greece, April 2006.

[41] Henan Zhao and Rizos Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *Proceedings of the 15th Heterogeneous Computing Workshop (HCW)*, Rhodes Island, Greece, April 2006.