MAHEFT-based Adaptive Grid Workflow Scheduling Approach

Ahmed A. Ghanem
Computers and Systems Engineering Department
Mansoura University
Mansoura, Egypt

Ahmed I. Saleh
Computers and Systems Engineering Department
Mansoura University
Mansoura, Egypt

Hesham A. Ali
Computers and Systems Engineering Department
Mansoura University
Mansoura, Egypt

ABSTRACT
The Grid Workflow scheduling is considered an important issue in Workflow management. Workflow scheduling is a process of assigning workflow tasks to suitable computational resources. Workflow scheduling significantly affects the performance and the execution time of the workflow. A Workflow scheduling approach falls in one of three categories: static, dynamic or adaptive. Grid environment is a highly changing environment in which static approaches performance is questioned. Effective workflow scheduling approaches are essential to make use of the Grid heterogeneous resource capabilities. The main objective of this paper is to introduce an adaptive heuristic list scheduling approach which utilizes the MAHEFT algorithm. MAHEFT algorithm considers the new changes in the Grid environment in order to minimize the total execution time (makespan) and to increase the speedup. The improvement rate in makespan of MAHEFT algorithm ranges between 2% to 21%. With respect to Speedup, MAHEFT is faster than both static HEFT and adaptive AHEFT algorithms with speedup values between 2.08 and 4.16.

General Terms
High Performance Computing, Grid Computing.

Keywords
Grid Workflow, Workflow Scheduling, DAG, Adaptive Scheduling, Makespan, Speedup.

1. INTRODUCTION
Science has greatly developed in the last three decades in a way that there is a need for high performance resources to solve big and complex problems. Grids are one of the solutions to this problem as Grids integrate large-scale distributed heterogeneous resources to enable users to access remote resources through secure and scalable networks. Many scientific fields such as high-energy physics, gravitational-wave physics, geophysics, astronomy, and bioinformatics are utilizing Grids to manage and process large data sets. Workflow applications [1] incorporate multiple dependent tasks to be executed in a predefined order and may entail the transfer and storage of a huge amount of data. The very important issue in executing a scientific workflow in Grids is how to map and schedule workflow tasks onto multiple distributed resources and handle task dependencies in a timely manner to deliver users' expected performance [2]. Workflow can be represented as a direct acyclic graph (DAG), in which nodes are the tasks and the edges are the inter-task dependencies. Each node label shows the task computation cost and each edge label shows the communication cost between tasks. The *makespan* is the total execution time of a workflow application, which is used to measure the performance of workflow applications.

Workflow scheduling is one of the key functions in the workflow management system [3]. Scheduling is a process that maps and manages the execution of inter-dependent tasks on the distributed resources. It allocates suitable resources to workflow tasks in order to achieve high performance. Proper scheduling can have significant impact on the performance of the system. According to [4], there are three types of workflow scheduling: *full-plan-ahead scheduling, Just-in-time scheduling* and *adaptive scheduling.* The first approach is a static one in which the whole workflow tasks are scheduled prior to the execution phase. In contrast, the second one is completely dynamic as it postpones the scheduling decision for a workflow task, as long as possible, and performed when the task is ready to be executed. The third approach is a hybrid one that combines the two former approaches. Full-plan-ahead scheduling is represented in GridFlow [5] and Vienna Grid Environment [6]. DAGMan [7] and Taverna [8] support dynamic scheduling, and Pegasus [9] supports both. The static scheduling performs near optimal when the *meta-data* about the Grid resources is known in advance. This meta-data includes the performance of task execution and data communication which is supposed to be accurate. But, it is not true for the Grid. Furthermore, static approaches are proven to perform better than dynamic ones even with inaccurate meta-data [10].

However, due to the nature of the Grid environment, static scheduling may perform poorly because of the continuous change in the Grid environment and the fact that the Grid resources are not dedicated but they are shared between many users and all of them compete for using them. Many challenges may face the static scheduling of workflow application including events like: resources join or leave at any time; the resource performance may vary over time due to internal or external factors. We claim that the performance of the static scheduling can be improved by combining them with adaptive strategy which reschedules the remaining of the workflow tasks when the Grid changes (i.e. when some resources appear or disappear). Heterogeneous Earliest Finish Time (HEFT) [11] static scheduling algorithm has been selected and modified to suit the proposed hybrid scheduling strategy. In static scheduling, the planning phase is a one-time process which does not consider

the future change in the Grid after the execution has started. With the new approach, the Monitor will notify the planner if any run-time event which interests the planner (i.e., the former events). Rescheduling may be needed and the schedule may change to make use of the new resources and resubmitting the tasks that were scheduled to leaving resources to other ones in order to reduce the *makespan*. This requires the planning phase to be an event-driven process. The experiment results show the improvement by the new adaptive approach in makespan and speedup.

The main contributions of this paper are: (1) proposing an adaptive rescheduling approach, (2) evaluating the performance of the adaptive approach, and (3) studying how the adaptive approach performs better than static approach and the other adaptive approach when a resource change happens.
The remainder of this paper is organized as follows. The next section presents related work. Problem definition is demonstrated in section 3. Then we describe the proposed scheduling approach in section 4. Section 5 indicates the experiments results and the evaluation of the proposed scheduling approach performance. Summary and Future work is given in section 6 of this paper.

2. RELATED WORK

In general, the problem of mapping workflow tasks on distributed resources belongs to a class of problems known as NP-hard problems [12], and because of its importance on performance it has been extensively studied and many heuristics were proposed in the literature. Heterogeneous Earliest Finish Time (HEFT) [11] is one of the most popular heuristics. It is implemented in ASKALON on WIEN2K application. Other heuristics such as Min-min, Max-Min [13], Critical-Path-on-a-processor (CPOP) [11] and Levelized Min Time (LMT) [14] are studied exhibiting the same strengths and weaknesses differing by few percent. HEFT is selected in this paper to implement the adaptive scheduling algorithm.

Static approaches used in Grid workflow applications have some challenges that are discussed in [15]. The hybrid approach proposed in [9] combines the just-in time scheduling and the full-ahead planning by partitioning the workflow into sub-workflows and by performing full-graph scheduling of the individual sub-workflows in a just-in-time manner. Adaptive Heterogeneous Earliest Finish Time (AHEFT) is another hybrid approach presented in [10]. AHEFT achieves the same goal by triggering rescheduling when the state of the Grid changes (i.e., when some resources appear or disappear). Rescheduling of applications is the most widely used method to make full-ahead planning more dynamic. To trigger rescheduling of an application, certain acceptance criteria defined for the application execution are needed, as well as a monitoring system which can control the fulfillment of these criteria.

Another rescheduling policy is proposed in [16], which considers rescheduling at a few, carefully selected points during the execution. The research tackles one of the shortcomings that static scheduling always assumes accurate prediction of task performance. After the initial schedule is made, it selectively reschedules some tasks if the run time performance variance exceeds predefined threshold. However, this approach deals with only the inaccurate estimation and does not consider the change of resource pool. Paper [17] describes how adaptive workflow execution can be expressed as an optimization problem where the objective of the adaptation is to maximize some property expressed as a utility function. It evaluates using an adaptive approach for alternative utility measures based on response time and profit.

We will focus on how the planner will adapt to the change in the Grid resources and if the new resources can be utilized to achieve higher performance and minimized makespan.

3. PROBLEM DEFINITION

Even though theoretically static scheduling performs near optimal [18], its effectiveness in a dynamic Grid environment is questioned. The next subsection discusses and analyzes the issues with static scheduling. Then, the new improvements that have been added to the system are presented. Next, the system architecture and its main components will be explained.

3.1 Issues with Static Scheduling

In static scheduling, planning is a one-time process which does not consider the future change of Grid environment after the resource mapping is done. Rescheduling in the execution phase is used to support fault tolerance. The overall issues with static scheduling are: (1) low accuracy of estimating the communication and computation costs of a DAG which is the key success factor of the scheduling strategy, (2) no adaptation to dynamic environment as the static scheduling approaches assume that the resource set is fixed over time. This assumption is not always valid, which results in low utilization for those resources that join after the plan is made.

3.2 Proposed System Improvements

The proposed scheduling strategy made planning an event-driven process rather than a one-time process by adding a *Monitor* component. The *Monitor* works collaboratively with the *Planner* and the *Executer* to make the system aware of any Grid environment change, including the task performance and resource availability. Another component has been also added, which is the *Intermediate Results Repository*. It stores the intermediate results of the running tasks in parallel with their execution. It allows starting the rescheduled task from the point it had reached before rescheduling.

3.3 Proposed System Architecture

The general architecture of the proposed system which adapts the planner to dynamic Grid environment is shown in Fig. 1. The system design, used by Yu *et al*. in [10], is considered with some modifications and improvements. The system consists of three main components: *Planner*, *Executer*, and *Monitor*. The GRID Services on top of which the system is built are essential services for any Grid system and are out of the scope of this paper. The role of each system component is defined as follows:

Planner. The *Planner* has a set of subcomponents including: *Scheduler*, *Performance History Repository* and *Predictor*. For each workflow application represented as a DAG, the *Planner* instantiates a *Scheduler* instance. The *Scheduler* inquires the *Predictor* to estimate the computation and the communication cost with the given resource set based on the performance history. Then, it decides an initial resource mapping in order to achieve the optimal performance of the entire workflow and

submits the schedule to the *Executer*. The initial resource mapping is made by first prioritizing the workflow tasks according to their dependencies. Then, the *Scheduler* selects the tasks with the highest priority to be scheduled on the resource that will achieve earliest finish time (EFT).

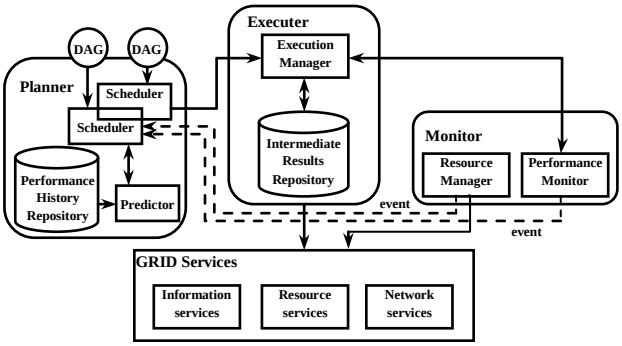


Fig 1: Proposed system architecture

Because of the Grid dynamic nature, the initial schedule may not be the optimal schedule. Adaptivity is achieved here by using rescheduling in order to adapt to the changes in the Grid resources. During the execution phase, the *Monitor* notifies the *Scheduler* with events that trigger rescheduling such as; (1) resource joins or leave or (2) change in the resource capabilities. *Scheduler* may decide a new schedule to make use of the new available resources if it minimizes the makespan of the workflow.

Executer. The *Executer* is an enactment environment for workflow applications. It consists of: *Execution Manager* and *Intermediate Results Repository*. *Execution Manager* receives a DAG schedule and executes it. It is also responsible for storing the output of each executed task to the *Intermediate Results Repository* to be ready for executing the child tasks on the mapped resource. If the schedule is a result of rescheduling, it stores the progress and the intermediate results of the running task that will be scheduled to a new resource such that the task can start executing from the end point it had reached before the reschedule has been made.

Monitor. The *Monitor* consists of two main components: *Resource Manager* and *Performance Monitor*. We have separated the *Monitor* from the executer to decrease the system complexity. The role of *Manager* and *Performance Monitor* is to update the Scheduler with the events such as:

- *Resource pool change.* The *Resource Manager* uses the GRID Services for discovering the resource pool changes. If new resources are discovered after the initial schedule, rescheduling may reduce the makespan of the workflow. In case of resource removal, fault tolerance mechanism is triggered and the *Execution Manager* takes care of it. *Execution Manager* stores the intermediate results of the running task on the leaving resource. It updates the *Performance Monitor* which notifies the *Scheduler* in turn. Using this mechanism, allows scheduling the remaining of the task only to another resource and no need to repeat the whole task.
- *Resource performance variance*. The performance estimation accuracy is largely dependent on history data. An inaccurate estimation leads to a bad schedule. If the run-time *Performance Monitor* can notify the *Planner* of any significant performance variance, the *Planner* will evaluate its impact and reschedule if necessary. In the meantime, the *Performance History Repository* is updated to improve the estimation accuracy in the subsequent planning.

4. HEFT-BASED ADAPTIVE SCHEDULING MAHEFT

The adaptive scheduling algorithm is described in Fig. 2. For a given DAG, an initial schedule is made. When there is a new resource available, the resource set is updated and the *Planner* tries to reschedule a randomly chosen running task.

MAHEFT scheduling algorithm

INPUT

Abstract workflow, *T* - set of tasks in the DAG, *R* - set of all available resources, *H* - Heuristic employed by scheduler.

OUTPUT

S - Schedule plan of the workflow tasks to Grid resources, and decide whether to reschedule the running task n_k to a new resource.

STEPS

- set initial schedule S_0 = *schedule (T, R, H)*
- **while** (resource pool change)
  - **do**
    # *R* is updated via communication with *Resource Manager*
    - update the resource set *R*

    # randomly choose n_k as one of the running tasks
    - choose task n_k

    # decide either to reschedule task n_k or not
    - **if** ($n_k.EFT_1 < n_k.EFT_0$)
      - store the intermediate results of task n_k and schedule the remaining part to the new resource

      # new schedule is made for the not started tasks only
      - S_1 = *schedule (T, R, H)*
    - **end if**
- **end while**

Fig 2: Adaptive scheduling algorithm (MAHEFT)

The remaining part of the chosen task will not be scheduled to the new resource unless its earliest finish time according to Equation (4) is less than the earliest finish time of the initial schedule. When scheduled to the new resource, the *Executer* will save the intermediate results to the *Intermediate Results Repository* and sends them to the new resource. Then, the *Planner* schedules the remaining tasks of the DAG. The *Planner* will continuously listen to an event from the Monitor to adapt to the Grid environment changes.

Next we define our scheduling strategy, which is HEFT-based scheduling algorithm, referred to as Modified Adaptive

Heterogeneous Earliest Finish Time (MAHEFT). We use HEFT to implement the *schedule (T, R, H)* method in the scheduling algorithm in Fig. 2. We directly use the scheduling system model defined in paper [11] with revision and extension. The input for a workflow scheduling algorithm is an abstract workflow which is a group of workflow tasks without allocating them to specific resources. A Workflow application is represented by a Directed Acyclic Graph (DAG), $G = (V, E)$, where V is the set of v tasks (nodes) and E is the set of e edges. Each edge $e\ (i, j) \in E$ represents precedence constraint such that task n_i should complete its execution before task n_j starts (i.e., n_i is the parent of n_j and n_j is the child of n_i). In any given task graph, a task without any parent is called an *entry task* and a task without any child is called an *exit task*. Also, we refer to *data* is a $v \times v$ matrix of communication data, where $data_{i,k}$ is the amount of data required to be transmitted from task n_i to n_k. R is a set of r heterogeneous resources which represent computation units connected in a fully connected topology in which all inter-processor communication are assumed to be performed without contention.

We define the symbols used by MAHEFT in Table 1, and explain how they are calculated according to the following equations. For the entry task, the EST is calculated according to Equation (1).

$$EST(n_{entry}, r_j) = 0 \quad (1)$$

For the other tasks in the graph, the EST and EFT are computed recursively, as shown in Equation (2) and (3) respectively. In order to compute the EFT of a task n_i, all immediate predecessor tasks of n_i must have been scheduled.

$$EST(n_i, r_j) = \max\{avail[j], \max_{n_m \in pred(n_i)}(AFT(n_m) + c_{m,i})\} \quad (2)$$

$$EFT(n_i, r_j) = \omega_{i,j} + EST(n_i, r_j) \quad (3)$$

The inner max block in the Equation (2) returns the ready time, i.e., the time when all the data needed by n_i has arrived at resource r_j. After task n_m is scheduled to resource r_j, the earliest finish time of n_m on resource r_j, is equal to the actual finish time, $AFT\ (n_m)$. The new EFT of the randomly selected task is calculated according to Equation (4). If the new *EFT* is less than the *EFT* resulted from the initial schedule, the remaining part of the chosen task is scheduled to the new resource. Equation (4) consists of three terms; the first term is the computation cost of the remaining part of the task n_i on the new resource r_k, the second term is the communication cost needed to transfer the output of the first part of the task n_i to the new resource r_k, and the third part is the point time of the reschedule. The second term is considered a penalty to reschedule the task to another resource.

$$EFT_1(n_i, r_k) = A + B + C \quad (4)$$

Table 1. Definition of symbols in MAHEFT

Symbol	Definition
$EST\ (n_i, r_j)$	the earliest start time for not-started task n_i on resource r_j
$EFT\ (n_i, r_j)$	the earliest finish time for not- started task n_i on resource r_j
$AFT\ (n_i)$	the actual finish time of task n_i
$avail[j]$	the earliest time when resource r_j is ready for executing new tasks
$\omega_{i,j}$	the average computation cost of task n_i on resource r_j
ϖ_i	the average computation cost of task n_i
$c_{i,j}$	the communication cost for data dependence of task n_j on task n_i
$\overline{c_{ij}}$	the average communication cost of edge (i, j)
$\overline{c_i}$	the average communication cost of task n_i
$succ(n_i)$	the immediate successors of task n_i
$pred(n_i)$	the set of immediate predecessor tasks of task n_i
$rem_{i,j}$	the remaining time for task n_i on resource r_j

The first term which is the computation cost of the remaining part of task n_i on the new resource r_k which is proportional to the computation cost of the task n_i on the new resource r_k and the remaining time for task n_i on resource r_j. It is inversely proportional to the computation cost of the task n_i on the new resource r_j. Where;

$$A \propto \omega_{i,k}$$

$$A \propto rem_{i,j}$$

$$A \propto \frac{1}{\omega_{i,j}}$$

Then A can be calculated as follows:

$$A = K_1 \frac{rem_{i,j} \times \omega_{i,k}}{\omega_{i,j}}$$

The second term which is the communication cost needed to transfer the output of the first part of task n_i to the new resource r_k is proportional to the time for completing the first part of task n_i on resource r_j and the average communication cost of task n_i. It is inversely proportional to the computation cost of the task n_i on the new resource r_j. Where;

$$B \propto (\omega_{i,j} - rem_{i,j})$$

$$B \propto \overline{c_i}$$

$$B \propto \frac{1}{\omega_{i,j}}$$

Then B can be calculated as follows:

$$B = K_2(\frac{\omega_{i,j} - rem_{i,j}}{\omega_{i,j}}) \times \overline{c_i}$$

The average communication of first part of task n_i is calculated by summing the communication cost between task n_i and all of its predecessor tasks divided by the number of the predecessor tasks as indicated in Equation (5).

$$\overline{c_i} = \sum_{n_j \in pred(n_i)} c_{i,j} / |pred(n_i)| \quad (5)$$

Assuming that constants K_1, K_2 are equal to 1, so the final Equation of the earliest finish time of the chosen task will be:

$$EFT_1(n_i, r_k) = \frac{rem_{i,j} \times \omega_{i,k}}{\omega_{i,j}} + (1 - \frac{rem_{i,j}}{\omega_{i,j}}) \times \overline{c_i} + C$$

After all tasks in a graph are scheduled, the schedule length (i,e., makespan) will be the actual finish time of the exit task n_{exit} as defined in Equation (6).

$$makespan = \max\{AFT(n_{exit})\} \quad (6)$$

The objective function of the workflow scheduling problem is to determine the assignment of tasks of a given application to resources such that its makespan is minimized.

It is obvious that MAHEFT is identical to HEFT when it is the initial schedule, (see Fig. 3). Tasks are ordered in the *schedule* (*T*, *R*, *H*) by their priorities based on *upward rank* of a task n_i is recursively defined by Equation (7)

$$rank_u(n_i) = \overline{\omega_i} + \max_{n_j \in succ(n_i)} (\overline{c_{ij}} + rank_u(n_j)) \quad (7)$$

Since the rank is computed recursively by traversing the task graph upward, starting from the exit task, it is called *upward rank*. The first term of the equation can be calculated according to Equation (8). The sum of the computation cost of task n_i on every resource is divided on the number of the resources. For the exit task, the upward rank is defined in Equation (9).

$$\overline{\omega_i} = \sum_{j=1}^{r} \omega_{i,j} / r \quad (8)$$

$$rank_u(n_{exit}) = \overline{\omega_{exit}} \quad (9)$$

As illustration, we use a sample DAG and resource set, shown in Fig. 4, to compare schedule performance of traditional HEFT, AHEFT and MAHEFT. The adjacent table shows the computation cost for each task on each resource. Resources r_1, r_2 and r_3 are available from the beginning while r_4 emerges at 13.

First, the tasks ranks are calculated starting from the exit task as in the previous example, the task ranks are calculated to be as shown in Table 2. For the entry task (n_1), the resource that will be chosen for executing it will be r_3 because it will give the earliest finish time equal to 9.

Procedure *schedule* (T, R, H) of MAHEFT

INPUT

T - set of the tasks of sate not started in the DAG, *R* - set of all available resources, *H* - HEFT heuristic employed by scheduler.

OUTPUT

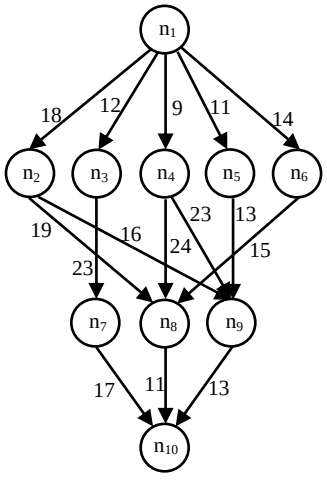
Computes the tasks upward rank $rank_u(n_i)$, and computes earliest finish time of each task on every resource to choose the more suitable resource with minimum earliest finish time.

STEPS

- compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task
- sort the tasks in a scheduling list by nonincreasing order of $rank_u$
- **while** (there are unscheduled tasks in the list)
  - **do**
    - select the first task n_i from the scheduling list
    - **for** each resource r_k in *R*
      - **do**
        - compute $EFT(n_i, r_k)$
    - assign task n_i to the resource that minimizes *EFT* of task n_i
- **end while**

Fig 3: Procedure schedule (T, R, H) of MAHEFT

The next task to be scheduled is task n_3. For tasks with one parent only such as task n_3, its *EST* will be either 9 if scheduled to resource r_3 or 9 + 12 = 21, which is the *EST* of task n_1 in addition to the communication cost between n_1 and n_3, if scheduled to any resource other than r_3. The *EFT* of the task n_3 is the basis for choosing the resource it will be allocated to.



Computation Cost

Task	Resource			
	r_1	r_2	r_3	r_4
n_1	14	16	9	14
n_2	13	19	18	17
n_3	11	13	19	9
n_4	13	8	17	15
n_5	12	13	10	14
n_6	13	16	9	16
n_7	7	15	11	15
n_8	8	11	7	20
n_9	18	12	20	13
n_{10}	21	7	16	15

Fig 4: A sample DAG, the weight of each edge represents its communication cost

The *EFT* of task n_3 on the three available resources r_1, r_2, and r_3, at the time point 9, are calculated to be 21 + 11 = 32, 21 + 13 =

34, and 9 + 19 = 28 respectively. So, the resource r_3 is chosen for executing task n_3. Resources are chosen for the tasks such as n_2, n_4… and n_6 will be chosen in the same way.

Table 2. Tasks upward ranking and selected resources

Task	Ranking	Resource
n_1	108	3
n_3	80	3
n_4	80	2
n_2	77	1
n_5	69	3
n_6	62	2
n_9	44.333	2
n_7	42.667	3
n_8	34.333	1
n_{10}	14.667	2

For tasks with more than one parent such as task n_9, the ready times of the three resources r_1, r_2, and r_3 are calculated to be 40, 42, and 49 which are the actual finish time of the tasks n_2, n_6, and n_7. The actual finish time of the predecessors of task n_9 which are task n_2, n_4, and n_5 are calculated to be 40, 26, and 38 respectively. For each resource, the maximum of the ready times of the three resources and the actual finish time of the predecessors in addition to the communication cost needed will be the *EST* of task n_9. So, if task n_9 is scheduled to resource r_1, the *EST* will be max (40, max (40 + 0 = 40, 26 + 23 = 49, 38 + 13 = 51)) which will be equal to max (40, 51) = 51. If task n_9 is scheduled to resource r_2, the *EST* will be max (42, max (40 + 16 = 56, 26 + 0 = 26, 38 + 13 = 51)) which will be equal to max (42, 56) = 56. If task n_9 is scheduled to resource r_3, the *EST* will be max (49, max (40 + 16 = 56, 26 + 23 = 49, 38 + 0 = 38)) which will be equal to max (49, 56) = 56. The *EFT* of task n_9 on the three resources will be 51 + 18 = 69, 56 + 12 = 68, and 56 + 20 = 76 respectively. The resource r_2 has the minimum execution time so it is chosen for executing task n_9. The chosen resources for executing the tasks are indicated in Table 2.

Fig. 5(a) shows the schedule obtained from HEFT that produces the schedule with makespan as 83 without considering the addition of resource r_4 at later time.

For AHEFT and MAHEFT, the initial schedule made at time point 0 is identical as the one by HEFT. When resource r_4 is added, the only task that is completed is n_1 and the task n_3 is scheduled on resource r_2. AHEFT considers rescheduling for only the non started tasks while, MAHEFT considers task n_3 also. HEFT produces schedule with makespan equal to 78 as shown in Fig. 5(b). The *EFT* of task n_3 on the new resource r_4 is calculated according to Equation 3.7, it is 25 which is less than the initial *EFT* equal to 36. So, task n_3 will be resumed on resource r_4 and the other tasks are then rescheduled as before in HEFT. The other non-started tasks are then ranked as illustrated before and they are allocated to the resources that have minimum *EFT*. The MAHEFT will reduce the schedule to 75 as shown in Fig. 5(c).

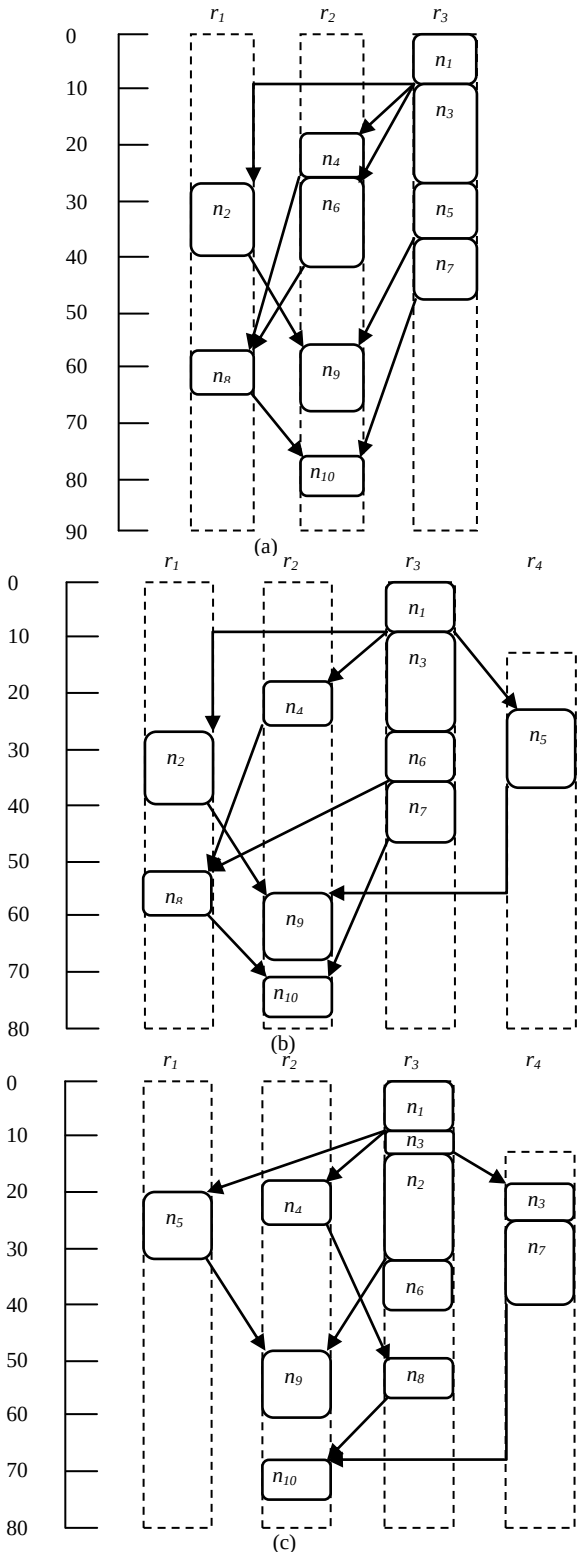


Fig 5: Schedule of the DAG in Fig. 4 using HEFT, AHEFT, and MAHEFT algorithms: (a) HEFT schedule (makespan=83), (b) AHEFT schedule with resource adding at time 13 (makespan=78), (c) MAHEFT schedule with resource adding at time 13 (makespan=75)

5. EXPERIMENT RESULTS

To evaluate the performance of the proposed MAHEFT scheduling algorithm we needed repeated cases of workflows and resource pools. A simulation of the framework will be introduced, in this section, to evaluate the proposed scheduling algorithm. The first subsection describes the metrics used for performance evaluation. The second subsection describes randomly generated DAGs and the parameters used to generate them.

5.1 Comparison Metrics

The comparison of the algorithms is based on the following two metrics:

Makespan. The main performance measure of a scheduling algorithm on a graph is the total execution time (makespan) of its output schedule.

Speedup. The speedup value for a given graph is computed by dividing the sequential execution time (i.e., cumulative computation costs of the tasks in the graph) by the parallel execution time (i.e., the makespan of the output schedule). The sequential execution time is computed by assigning all tasks to a single processor that minimizes the cumulative of the computation costs. The speedup is defined by Equation (10)

$$Speedup = \frac{\min_{r_j \in R} \{\sum_{n_i \in V} w_{i,j}\}}{makespan} \quad (10)$$

5.2 Randomized Generated DAGs

The Standard Task Graph Set (STG) [19] is a kind of benchmark for evaluation of multiprocessor scheduling algorithms. STG is used to evaluate the algorithms under the same conditions covering various DAGs generation methods including DAGs generated from actual application programs. The STG consists of two sets of DAGs, the first set contains DAGs generated from actual application programs while, the second set contains 900 randomly generated DAGs in which DAG size varies between 50 and 2700 tasks. DAG shapes (precedence constraints) are determined based on four different methods [20-22].

A random DAG generator was implemented to generate different DAGs with various characteristics based on some input parameters. For fair comparison with the static HEFT and the adaptive AHEFT algorithms, the generator uses the same values of the input parameters used in the approach utilized in [11] to determine the DAG shape. These input parameters are also suggested in the workflow test bench work [23]. The generator follows the fourth method of "*layrpred*" but, using different number of layers and the number of tasks in each layer. The generator will use the following set of parameters:

- The number of tasks in the graph (υ).
- Shape parameter of the DAG, (α). The height of the DAG (no. of layers) is calculated according to Equation (11).

$$\text{Number of layers} = \frac{\sqrt{\upsilon}}{\alpha} \quad (11)$$

- Out degree of a node, (*out_degree*). The width of each layer (number of tasks) is generated randomly such that it cannot exceed the number of tasks in the previous level multiplied by the out degree of a single node. A dense DAG i.e., a shorter DAG with high degree of parallelism can be generated by choosing $\alpha >> 1$ because the number of layers is inversely proportional to α . While, a longer DAG with high number of layers can be generated if $\alpha << 1$.
- Communication to computation ratio (*CCR*). It is the ratio of the average communication cost to the average computation cost. A data-intensive application has a higher *CCR*, while a computing-intensive one has a lower value of *CCR*.
- The resource heterogeneous factor, β. A higher value of β suggests the bigger difference of resource capability. The resources are homogeneous when β is 0. The average computation cost of all tasks in a DAG is $\overline{\omega_{DAG}}$, then the average of each task n_i in the graph, represented as ϖ_i, is selected randomly from a uniform distribution with range [0, $2 \times \overline{\omega_{DAG}}$]. Then, the computation cost of each task n_i on each resource r_j in the system, i.e., $\omega_{i,j}$, is randomly selected from the following range: $\varpi_i \times (1-\frac{\beta}{2}) \le \omega_{i,j} \le \varpi_i \times (1+\frac{\beta}{2})$.

The set of values for each of the previously stated parameters is given in Table 3. These combinations results in 1875 different DAGs.

Table 3. Value set of random generated DAGs parameters

Parameter	Value
υ	20, 40, 60, 80,100
α	0.5, 1.0, 2.0
out_degree	1, 2, 3, 4, 5
CCR	0.1, 0.5, 1.0, 5.0, 10.0
β	0.1, 0.25, 0.5, 0.75, 1.0

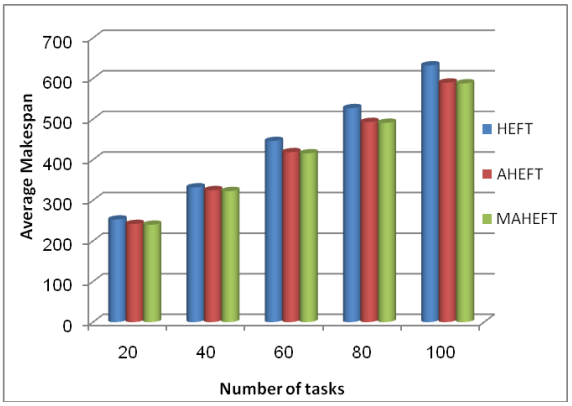
To model the dynamic change of resources, we introduce three additional parameters as follows: (1) Initial resource pool size, *R*; (2) Interval of resource change, ω. The higher value of ω indicates the lower frequency of resource change; and (3) Percentage of resource change, Δ, to measure the resource change percentage each time compared with the initial resource pool. The value set for each of these parameters is listed in Table 4.

Table 4. Value set of additional parameters

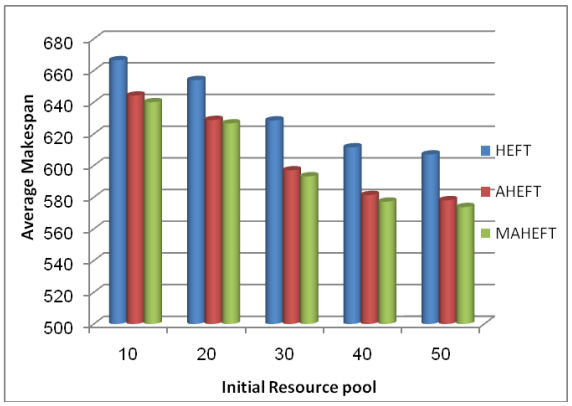
Parameter	Value
R	10, 20, 30, 40, 50
Δ	0.1, 0.15, 0.2, 0.25
Ω	50, 100, 200, 400

To study the effect of the previously stated parameters on the makespan and speedup, seven experiments are used each with 50 random DAGs generated using the implemented randomized DAG generator.

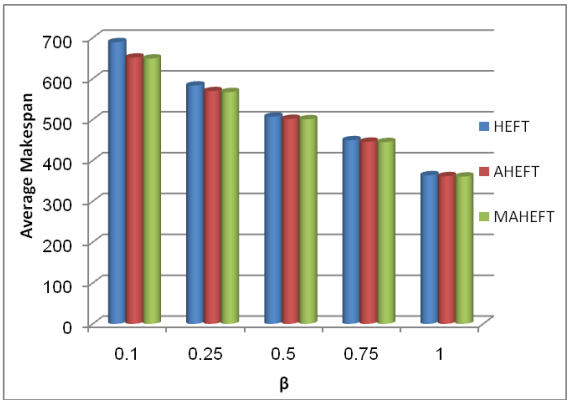
Fig. 6 presents the results of HEFT, AHEFT and MAHEFT for randomized generated DAGs. In Fig. 6(a), the makespan increases with total number of tasks because more tasks means more computation and communication cost needed. In Fig. 6(e), the makespan of the three algorithms tends to be the same with larger values of resource change interval because the higher the resource change interval the more static become the Grid environment and this leads to no change in it.



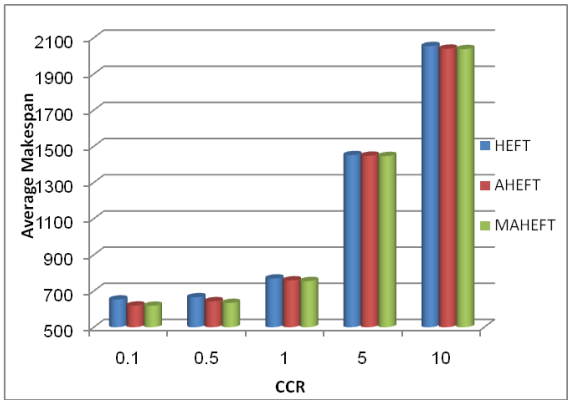
a) Makespan at different number of tasks.



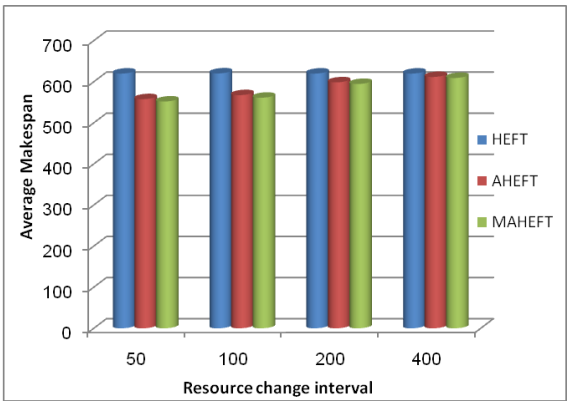
b) Makespan at different Initial Resource pool.



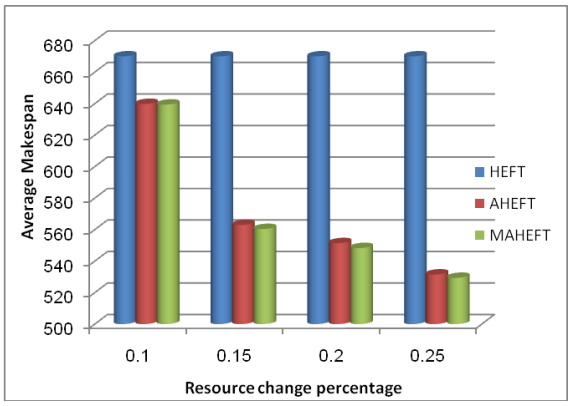
c) Makespan at different heterogeneity factor (β).



d) Makespan at different Communication-to-Computation Ratio (CCR).



e) Makespan at different Resource change interval.



f) Makespan at different Resource change percentage.

Fig 6: Average makespan at different parameters values

Fig. 6(f) shows the effectiveness of our algorithm more clear when the resource change percentage increases because if the new added resources were better than those in the initial pool there will be more benefit of using them.

To indicate how the adaptive algorithms AHEFT and MAHEFT perform better than the HEFT static algorithm, the improvement rate in makespan is used here. Fig. 7 shows the relationship of the improvement rate in makespan of both algorithms with the resource change interval. It worth noting that with increasing the resource change interval the improvement rate of the two adaptive algorithms decreases because the environment is more static and the changes are slower.

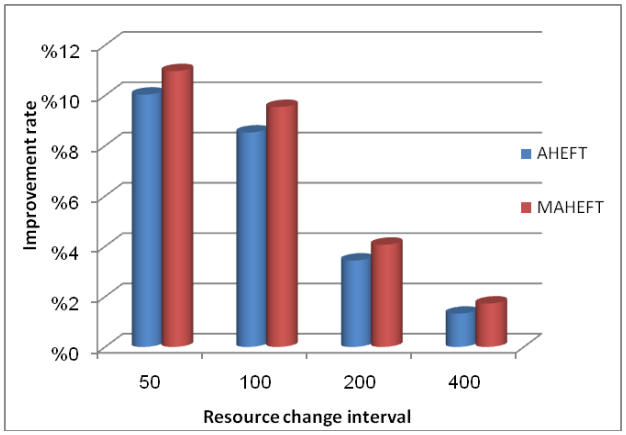


Fig 7: Improvement rate at different Resource change interval.

Fig. 8 shows that the improvement rate in makespan increases with higher values of Δ as when better resources join the resource pool they can be used to execute DAG tasks.

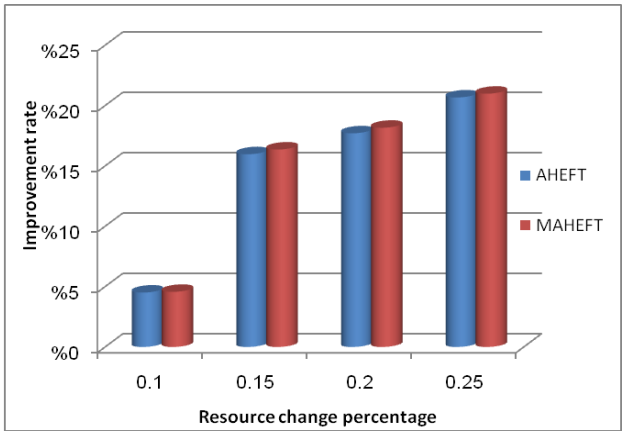


Fig 8: Improvement rate at different Resource change percentage.

The speedup of a workflow scheduling algorithm is another performance metric to be considered. This experiment indicates the speedup at different total number of tasks in the DAG. Table 5 lists the speedup values of the three algorithms with varying the number of tasks. It is observed that with the increase in the total number of tasks, the speedup jumps initially and becomes stable later.

Table 5. Speedup at various total number of tasks.

Number of tasks	20	40	60	80	100
HEFT Speedup	1.97	2.8	3.27	3.67	3.75
AHEFT Speedup	2.07	2.86	3.49	3.93	4.02
MAHEFT Speedup	2.08	2.96	3.51	3.93	4.16

6. SUMMARY AND FUTURE WORK

This paper analyzes issues of static scheduling strategy for grid workflow applications, and proposes an adaptive scheduling strategy. The new approach exploits its inherent benefits. MAHEFT is developed and tested for its stability and effectiveness with various DAGs, and the results are promising. The analysis of the experiments shows that the MAHEFT outperforms both the static HEFT and the adaptive AHEFT algorithms. The improvement rate in makespan of the MAHEFT algorithm ranges between 2% to 21% according to the DAG type and the Grid environment parameters. It also shows that MAHEFT prefers computation-intensive applications. With respect to the Speedup, the MAHEFT is faster than both HEFT and AHEFT algorithms with speedup values between 2.08 and 4.16. It worth noting that, our MAHEFT algorithm performs better than the other two algorithms. The improvement rate may vary according to the DAG type generated.

7. REFERENCES

[1] I. Taylor, E. Deelman, D. Gannon, and M. Shields. Workflows for e-Science: Scientific Workflows for Grids. Springer, 2006.

[2] I. Foster and C. Kesselman. Computational Grids in The Grid: Blueprint for a New Computing Infrastructure. Springer, ch. 2, 1999.

[3] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, Springer, Sept. 2005.

[4] M. Wieczorek, R. Prodan, A. Hoheisel, M. Wieczorek , R. Prodan, and A. Hoheisel. Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem. Grid Middleware and Services Book. p 237-264. Springer US, 2008.

[5] J. Cao, S. Jarvis, S. Saini, and G. Nudd. GridFlow: Workflow Management for Grid Computing. In 3rd International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan, IEEE Computer Society Press, Los Alamitos, May 12-15, 2003.

[6] I. Brandic, S. Pllana, and S. Benkner. Amadeus: A Holistic Service-oriented Environment for Grid Workflows. International Workshop on Workflow Systems in Grid Environments (WSGE06). China, October 2006.

[7] J. Frey and et al., Condor-g: A computation management agent for multi-institutional grids. Cluster Computing Journal, 237–246, July 2002.

[8] T. Oinn and et al., Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinfomatics, 3045–3054, 2004.

[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, et al. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal, Nov. 2005.

[10] Z. Yu and W. Shi. An Adaptive Rescheduling Strategy for Grid Workflow Applications. In Proceedings of the 21st IPDPS, 2007.

[11] H. Topcuouglu, S. Hariri, and M.-Y. Wu. Performance effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distribution Systems, 260–274, 2002.

[12] J. D. Ullman, NP-complete Scheduling Problems, Journal of Computer and System Sciences, 384-393, 1975.

[13] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems. In 8th Heterogeneous Computing Workshop (HCW'99), Apr. 1999.

[14] Iverson, M., F. Ozguner and G. Follen. Parallelizing existing applications in a distributed heterogeneous environments. Proc. Heterogeneous Computing Workshop, 93-100, 1995.

[15] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. Grid Resource Management, State of the Art and Future Trends, 99–116, 2004.

[16] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. Scientific Programming, 253–262, 2004.

[17] K. Lee, N. W. Paton, R. Sakellariou, A. Fernandes. Utility Driven Adaptive Workflow Execution. In the Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 220-227, 2009.

[18] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. IEEE International Symposium on Cluster Computing and Grid (CCGrid), 2005.

[19] STG, http://www.kasahara.elec.waseda.ac.jp/schedule/index.html, visited June 2011.

[20] V. Almeida, I. Vasconcelos, J. Árabe and D. A. Menascé, "Using Random Task Graphs to Investigate the Potential Benefits of Heterogeneity in Parallel Systems," Proc. Supercomputing '92, pp. 683-691, 1992.

[21] T. Yang and A. Gerasoulis, "DSC : Scheduling Parallel Tasks on an Unbounded Number of Processors," IEEE Trans. Parallel and Distributed Systems, Vol.5, No.9, pp. 951-967, 1994.

[22] T. Adam, K. Chandy and J. Dickson, "A Comparison of List Schedules for Parallel Processing Systems", Communications of the ACM, Vol.17, No.12, pp. 685-690, 1974.

[23] U. H¨onig and W. Schiffmann. A comprehensive test bench for the evaluation of scheduling heuristics. In roc. of PDCS 2004.