A Monte-Carlo Approach for Full-Ahead Stochastic DAG Scheduling

Wei Zheng
Department of Computer Science
Xiamen University
Xiamen, China
zhengw@xmu.edu.cn

Rizos Sakellariou
School of Computer Science
The University of Manchester
Manchester, U.K.
rizos@cs.man.ac.uk

***Abstract*—In most heterogeneous computing systems, there is a need for solutions that can cope with the unavoidable uncertainty in individual task execution times, when scheduling DAGs. When such uncertainties occur, static DAG scheduling approaches may suffer, and some rescheduling may be necessary. Assuming that the uncertainty in task execution times is modelled in a stochastic manner, then we may be able to use this information to improve static DAG scheduling considerably. In this paper, a novel DAG scheduling approach is proposed to solve this stochastic scheduling problem, based on a Monte-Carlo method. The approach is built on the top of a classic static scheduling heuristic and evaluated through extensive simulation. Empirical results show that a significant improvement on average application performance can be achieved by the proposed approach at a reasonable execution time cost.**

***Keywords*-Directed Acyclic Graph; full-ahead scheduling; DAG scheduling; monte-carlo methods;**

I. Introduction

As heterogeneous distributed computing systems (e.g., clusters, Grids, Clouds etc.) become common in order to cater to massive computational demands of running complex applications which comprise of multiple tasks, the process of assigning these tasks to multiple resources, known as scheduling, becomes significantly important to application performance. Among the different applications, Directed Acyclic Graphs (DAGs) have always received lots of attention. In recent years, this attention has increased as a result of the increased interest in scientific workflow applications, which are often modelled by DAGs [1].

In a DAG, nodes denote *tasks* and edges represent data transmission among tasks. Given a set of resources, a *schedule* for a DAG is an assignment which specifies the mapping of tasks and resources and the estimated start time of each task on the mapped resource. A typical aim of DAG scheduling is to minimize the overall execution time (i.e., *makespan* of the schedule). To achieve this aim is challenging, since the DAG scheduling problem has been proved to be NP-hard [2].

A key aspect in DAG scheduling is the use of estimates of the execution time for each task of the DAG on each available (heterogeneous) resource. Conventionally, such estimates are often assumed to be constant in the literature related to DAG scheduling in heterogeneous systems [3]–[5]. Then, a schedule is built based on a collection of such estimates. Such a schedule is called a '*static schedule*'. However, in practice, especially in dynamic environments (e.g. grids), such a static schedule seldom leads to good application performance since the task execution time is inherently unpredictable (i.e., the actual execution times of tasks at run-time usually deviate from their static predictions [6]). Thus, the question that arises is how to tackle the unpredictability in order to achieve good performance for DAG applications in dynamic computing environments. Generally, there are two main DAG scheduling approaches to deal with the unpredictability: *just-in-time scheduling* and *rescheduling*.

The just-in-time scheduling approach means that every task is scheduled only when it becomes ready (namely, all tasks that the current task depends upon have completed their execution). Based on this idea, some heuristics [7], [8] have been developed, which consider only the DAG structure in order to improve application performance. Nevertheless, the evaluation in [9] indicates that, without considering task execution costs when scheduling tasks, the practical effectiveness of these heuristics, at their current form, is limited.

In contrast, rescheduling suggests that an initial schedule for the DAG application is produced based on static prediction. Then, the allocated tasks are rescheduled according to an assessment of variations during run-time. In many cases, rescheduling can indeed improve application performance in comparison with the schedule generated based on static prediction. However, this approach may be costly as it has to introduce extra scheduling efforts during run-time. Thus, it may turn out to be a time-consuming solution even though some selective policies, aiming at reducing the number of rescheduling attempts, have been suggested [10].

This paper adopts the view that in addition to taking action during run-time, as the aforementioned approaches do, it is also important to craft a static schedule with good properties before the task execution starts (namely, a full-ahead schedule [11]). A well-crafted static schedule can, not only save the extra effort for rescheduling during run-time, but also obtain better application performance (to be

demonstrated later) in comparison with a badly-crafted static schedule even with rescheduling. Moreover, as indicated in relevant studies [10], a good initial schedule would help reduce the extra cost of rescheduling.

An abundance of DAG scheduling heuristics based on static prediction have been developed; for a comparative evaluation of several such heuristics we refer to [12]. None of these heuristics focuses on minimizing makespan in a dynamic context. In such context, one popular and straightforward way to express the dynamic nature of the environment is to model the predicted task execution times in a stochastic manner [13]–[18]; for example, one can assume that the execution time of every task in the DAG is modelled using a normal distribution (with specific values for mean and standard deviation). In this case, the DAG scheduling problem boils down to finding a full-ahead schedule that minimizes expected makespan (which can also be viewed as a random variable). In other words, assuming that task execution times follow a specific distribution, the schedule chosen should minimize the expected makespan; this is a *stochastic scheduling* problem. Intuitively, due to the task dependencies in a DAG, it is difficult to specify analytically the exact distribution that the makespan will follow, even if the distributions of task execution times are known. As a result, the best approach to solve the problem is by using a non-analytical method.

In this paper, a novel Monte-Carlo based DAG scheduling approach is proposed to generate a static schedule before run-time that minimizes the expected makespan. In the experimental section of this paper, it is shown that no matter if a rescheduling technique is applied or not, the proposed approach can effectively reduce the expected makespan in most cases without making the scheduling time cost prohibitively high.

The remainder of the paper is organized as follows. Related work is reviewed in Section II. The targeted scheduling problem and the associated assumptions are stated in Section III. A novel scheduling approach is presented in Section IV and illustrated by an example DAG provided in Section V. Section VI evaluates the effectiveness of the proposed approach. Finally, the paper is concluded in Section VII.

II. Background and Related Work

Dozens of static DAG scheduling heuristics aiming at minimizing makespan for heterogeneous systems have been presented in the literature. These heuristics are designed using different design principles and with different motivation. On this basis we can roughly classify these heuristics into: list scheduling (e.g., HEFT [3] including its variations [19], HBMCT [4], GDL(DLS) [20], PCT [21], CPOP [3], ETF [22], BIL [23]), workflow-based scheduling (e.g., WBA [24], ILS [25]), guided search scheduling (e.g., GA [26], SA [27]), clustering based scheduling(e.g., Triplet [28]), and task duplication based scheduling (e.g., TDS [29], STDS [30], LDBS [31]). An extensive list of references of DAG scheduling heuristics can also be found in [12].

Input: A DAG application G.
Output: A schedule for G.

Compute the weights of nodes and edges.
Compute bottom-level ranking for each node.
Sort all tasks in the descending order of bottom level (priority) and put them into list L.
while there are unscheduled tasks in L
  Select task i with the highest priority from list L.
  Compute $ft_{i,p}$ on each resource p.
  Allocate i to the resource p' that gives the minimum $ft_{i,p'}$ considering possible insertion.
  Remove i from list L.
endwhile

Figure 1: The HEFT Heuristic

Most of these heuristics consider the estimate of task execution time to be constant (or deterministic) and known in advance. Even when task execution time estimates are modelled by a probability distribution, one could still use one of the above-mentioned static heuristics by deriving a single value for task estimates (e.g., the mean of the probability distribution). However, as will be demonstrated in this paper, this approach does not lead to the best possible schedules in most cases. Thus, in this paper, we derive a DAG scheduling approach, which is based on Monte-Carlo principles, to take advantage of information about task execution times which can be provided as a probability distribution (as opposed to a single constant value). Although our proposed approach can work with any static DAG heuristic, a commonly cited, well-known heuristic is used: HEFT [3], which is chosen in our implementation and evaluation. A brief description of HEFT is provided in the next paragraph.

HEFT (Heterogeneous Earliest Finish Time) [3] is a static priority-based list scheduling heuristic which aims to minimize the makespan of DAG applications on a bounded number of heterogeneous resources. As described in Figure 1, the heuristic applies bottom-level ranking to prioritize tasks in the listing phase, and then, in turn, selects the resource which minimizes the estimated finish time of the given task in the scheduling phase. It is worth mentioning that, when estimating the finish time of a task on a resource (denoted by $ft_{i,p}$ in Figure 1), HEFT allows a task to be inserted into the existing task queue of the resource as long as task dependencies permit. The bottom-level ranking of a task i is defined below:

$$bLevel(i) = w_i + \max_{j \in Succ(i)} \{w_{i \to j} + bLevel(j)\} \quad (1)$$

where $Succ(i)$ means the set of all the immediate successors (i.e., child tasks) of task i, and w_i and $w_{i \to j}$ are, respectively,

weights of nodes and edges, computed by

$$w_i = (\sum_{p \in \mathcal{R}} et_{i,p})/(|\mathcal{R}|) \tag{2}$$

$$w_{i \to j} = (\sum_{p,q \in \mathcal{R}} tl_{(i,p) \to (j,q)})/(|\mathcal{R}| \cdot |\mathcal{R}|) \tag{3}$$

where $|\mathcal{R}|$ denotes the number of resources, $et_{i,p}$ the execution time of task i on resource p and $tl_{(i,p) \to (j,q)}$ the transmission latency between task i and j, which are separately allocated to resource p and q.

Due to the inaccuracy of task execution time prediction, the schedule generated by applying any full-ahead scheduling heuristic to the static prediction may make bad decisions during scheduling [32]. In the context of heterogeneous computing systems, there are a couple of scheduling studies, which consider a stochastic model for modelling task execution times. In [15] and [16], stochastic scheduling approaches based on a Genetic Algorithm were proposed to minimize the scheduling length. With the same objective, in [18], in addition to the mean value of the random weights of nodes, standard deviations are considered to extend the Min-Min and Max-Min algorithms to derive a schedule. In [33], where task execution times are also modelled stochastically, a set of greedy and iterative heuristics were proposed for robust static resource allocation. However, all of these approaches focus only on applications composed of independent tasks with no data dependencies, which differentiates our work in this paper, whose focus is DAGs.

For DAG applications, the robustness of a series of static scheduling heuristics was evaluated in [12] based on a stochastic model. Lòpez and Senar [34] analyzed the performance of several DAG scheduling heuristics and their counterpart dynamic version (with rescheduling) with a stochastic model. However, these studies do not address the issue of how to take advantage of a stochastic model to produce efficient full-ahead schedules for the DAGs. In [35], a DAG clustering approach based on a genetic algorithm is proposed to address the uncertainties caused by unstable data transmission when running DAG applications. In this work, the uncertainty of task execution time prediction is not considered. Canon and Jeannot [36] proposed a variety of strategies for the bicriteria problem of minimizing the makespan while maximizing the robustness for stochastic DAGs. In contrast, our work focuses on optimizing a single, well-defined criterion, makespan.

There have been efforts to construct analytically a full-ahead schedule for a stochastic DAG. In [17], two static DAG scheduling heuristics, ETF and DLS, were extended by estimating the earliest start times of tasks in a stochastic manner to minimize the scheduling length. This work introduced computation and comparison of random variables to the process of computing the earliest start time for a task to make a mapping decision. Similarly, in [37], HEFT is extended to make use of a stochastic DAG model. In these works, task execution times are assumed to follow a specific distribution (e.g., exponential) to ease the analysis.

Our work differentiates with existing approaches in the way that we generate a full-ahead DAG schedule when a stochastic model for task execution times is used. Instead of computing it analytically, we generate a full-ahead schedule via Monte-Carlo methods, which is a sound approach with two advantages: (a) it can avoid the complex computation involved with handling analytically random variables, and (b) it is applicable to any random distribution, as long as random sampling is possible.

III. Problem Description

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote a DAG which is a directed graph consisting of a set of nodes $\mathcal{N}$ and a set of edges $\mathcal{E}$, each of which is of the form $(i \to j)$, where $i, j \in \mathcal{N}$. A node i represents the counterpart task (the terms 'node' and 'task' will be used interchangeably hereafter), and an edge $i \to j$ denotes the inter-task dependency between task i and j. The execution of task j cannot begin until task i has finished and the required data transmission from i to j has been completed. Given an edge from i to j, i is called a parent node of j, and j a child of i. Parentless nodes are called *source nodes*; childless nodes are called *sink nodes*. Apparently, an entry node of $\mathcal{G}$ must be a source node, and an exit node a sink node. For standardization, we specify that a DAG has only a single entry node and a single exit node. It is easy to show that all DAGs with multiple entry or exit nodes can be equivalently transformed to this standardization.

It is assumed that there is a set $\mathcal{R}$ of multiple heterogeneous resources that are fully connected, and the time needed to transmit per unit of data from one resource to another, named *transmission latency*, is constant and pre-known as

$$tl_{(i,p) \to (j,q)} = d_{i \to j} \times \eta_{p,q} \tag{4}$$

where $d_{i \to j}$ is the size of transmitted data from task i to j, and $\eta_{p,q}$ is the transmission rate between resource p and q. It is also assumed that one resource can only run one task at a time and no preemption is considered.

The predicted execution time of task i on resource $p \in \mathcal{R}$ is modelled as a random variable $ET_{i,p}$. In a schedule for a DAG application, each assignment of task i to resource p is assigned a start time $ST_{i,p}$ and a finish time $FT_{i,p}$. In terms of the aforementioned constraint about the execution of one task of the DAG on a resource, it is clear that the following equation is satisfied

$$ST_{j,q} = \max\{FT_{l^*,q}, \max_{k \in Par(j)}\{FT_{k,r(k)} + tl_{(k,j) \to (r(k),q)}\}\} \tag{5}$$

where $FT_{l^*,q}$ represents the finish time of task l^* which is the currently last task on q, $Par(j)$ denotes the set of all parent tasks of j, $r(k)$ denotes the resource where k is

assigned to, and each task k in $Par(j)$ has been assigned to resource $r(k)$ with the earliest finish time $FT_{k,r(k)}$. Also, $tl_{(k,j)\rightarrow(r(k),q)}$ is defined in Eq.(4). Apparently,

$$FT_{j,q} = ST_{j,q} + ET_{j,q}. \tag{6}$$

In the case of an entry node, we have:

$$FT_{entry\_node,r(entry\_node)} = ET_{entry\_node,r(entry\_node)} \tag{7}$$

since the entry node is assumed to start at time 0. In addition, if there is no task assigned to resource q, $FT_{l^*,q} = 0$. Due to the randomness of task execution time, the makespan of the schedule, denoted by

$$M = FT_{exit\_node,r(exit\_node)}, \tag{8}$$

is also a random variable. Apparently, different schedules may result in different M's.

The scheduling problem to solve in this paper is to appropriately assign every task i of $\mathcal{G}$ onto a suitable resource m before run-time and generate a schedule Ω in order to minimize the expected value of makespan of Ω (denoted by $\overline{M}_\Omega$) based on task execution time predictions, which are modelled by a random variable following a probability distribution. It is also worth mentioning that, regardless of the actual task execution time, the task execution order on any local resource will remain consistent with the schedule.

IV. Methodology

The basic idea of the proposed approach is derived from the Monte-Carlo method [38], a popular solution for various problems which are infeasible or impossible to resolve by deterministic computation. The main characteristic of the Monte-Carlo method is that it normally utilizes repeated random sampling to obtain numerous random samples of a stochastic problem, and computes its results based on the obtained samples to resolve the problem. A common pattern of the Monte-Carlo method usually comprises the following steps:

1) define a space comprising possible input values;
2) take an independent sample randomly from the space;
3) perform a deterministic computation using the taken sample as input and store the result;
4) repeat Steps 2 and 3 until a pre-specified maximum number of repetitions is reached;
5) aggregate the stored results of the individual computations into the final result.

Our proposed Monte-Carlo based Scheduling approach (MCS hereafter) adapts the above-mentioned operations into a DAG scheduling scenario. Suppose that a DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is being scheduled on a set of resources $\mathcal{R}$; then, the adapted operations can be interpreted as follows:

- *Define Input Space.* The input space of MCS (denoted by $\mathcal{I}$) is a set consisting of the random task execution time predictions of the given application, namely

$$\mathcal{I}_\mathcal{G} = \{ET_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\} \tag{9}$$

- *Random Sampling.* In MCS, taking a sample from the space is defined as sampling each random prediction in the space and resulting in a set of static predictions. This can be expressed by function f_{smp} as below

$$\mathcal{P}_\mathcal{G} = f_{smp}(\mathcal{I}_\mathcal{G}) = \{t_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\} \tag{10}$$

  where $\mathcal{P}_\mathcal{G}$ can be viewed as a set of static task execution time predictions of $\mathcal{G}$, and $t_{i,p}$ is a random sampling of $ET_{i,p}$. Particularly, $\overline{\mathcal{P}_\mathcal{G}} = \{\mu_{i,p} : i \in \mathcal{N}, p \in \mathcal{R}\}$ is used to denote the estimation set consisting of the means of task execution time predictions, where $\mu_{i,p}$ is the expected value of $ET_{i,p}$.
- *Deterministic Computation.* There are two types of deterministic computations defined in MCS, one of which is to compute a static schedule. Given a DAG $\mathcal{G}$ and the associated set of static predictions $\mathcal{P}_\mathcal{G}$, any static scheduling heuristic $\mathcal{H}$ can be applied to generate a static schedule $\Omega_\mathcal{G}$, namely,

$$\Omega_\mathcal{G} = \textit{Static\_Scheduling}_\mathcal{H}(\mathcal{G}, \mathcal{P}_\mathcal{G}) \tag{11}$$

  More specifically, there is a *mean static schedule* which is obtained by applying $\mathcal{H}$ to $\overline{\mathcal{P}_\mathcal{G}}$, namely,

$$\overline{\Omega}_\mathcal{G} = \textit{Static\_Scheduling}_\mathcal{H}(\mathcal{G}, \overline{\mathcal{P}_\mathcal{G}}) \tag{12}$$

  The other type of computation is to calculate the static makespan of a generated schedule on the assumption that the exact execution time of each task on each resource is known as $\mathcal{P}^*_\mathcal{G}$. This computation can be denoted by

$$m^* = \textit{Calculate\_Makespan}(\mathcal{G}, \mathcal{P}^*_\mathcal{G}, \Omega_\mathcal{G}) \tag{13}$$

Generally, MCS consists of two main phases, namely, '*producing*' and '*selecting*'. In the producing phase, a considerable number of samples are taken from the Input Space and, accordingly, a long list of different static schedules is generated by employing a specific static scheduling heuristic (HEFT in our case, but it could be any). Again, in the selecting phase, a certain number of samples are taken to evaluate the generated schedules, which are then compared to obtain the selected schedule for output. Based on the above-defined equations, MCS is described in Figure 2.

It is worth explaining in Line 6 that a schedule Ω is QUALIFIED if

$$\overline{M}_\Omega < \overline{M}_{std} \cdot (1 + \Delta), \tag{14}$$

where $\overline{M}_{std}$ is the base of a threshold, taken as the makespan of the schedule produced when the mean task execution time is used. The Δ is a small positive real number, which acts as a variable weight to control the strictness of the threshold and can be determined empirically. The idea

Input: A DAG application $\mathcal{G}$ with stochastic performance prediction $\mathcal{I}_{\mathcal{G}}$.
Output: A full-ahead schedule $\Omega_{\mathcal{G}}$ for $\mathcal{G}$.

1: Create an empty schedule list $\mathcal{L}$.
2: Apply any DAG scheduling heuristic $\mathcal{H}$ to the mean of random task execution times so that the mean static schedule as defined in Eq.(12) is computed and put it into $\mathcal{L}$.
3: **while** the termination condition of the producing phase is not met **repeat**
4: Take a sample of the stochastic performance prediction as defined in Eq.(10), which results in a set of static task execution time predictions $\mathcal{P}_{\mathcal{G}}$.
5: Generate a static schedule $\Omega_{\mathcal{G}}$ by applying a static heuristic $\mathcal{H}$ to $\mathcal{P}_{\mathcal{G}}$, as defined in Eq.(11).
6: Add $\Omega_{\mathcal{G}}$ into $\mathcal{L}$ if $\Omega_{\mathcal{G}}$ is never produced before and is approved to be likely to obtain a good average makespan (i.e., QUALIFIED as explained in Section IV).
7: **endwhile**
8: **for** each repetition in selecting phase **do**
9: Take a sample of the stochastic performance prediction as defined in Eq.(10), which results in a set of static task execution time predictions $\mathcal{P}^*_{\mathcal{G}}$.
10: **for** each schedule $\Omega_{\mathcal{G}}$ in L **do**
11: Evaluate the makespan of $\Omega_{\mathcal{G}}$ by assuming that $\mathcal{P}^*_{\mathcal{G}}$ depicts the actual task execution times of $\mathcal{G}$ as defined in Eq.(13).
12: **endfor**
13: **endfor**
14: Compute the average makespan for each $\Omega_{\mathcal{G}}$ over the loops of evaluation.
15: **Return** the schedule with the minimum average makespan as the result schedule.

Figure 2: The description of our proposed Monte-Carlo approach for DAG scheduling, MCS

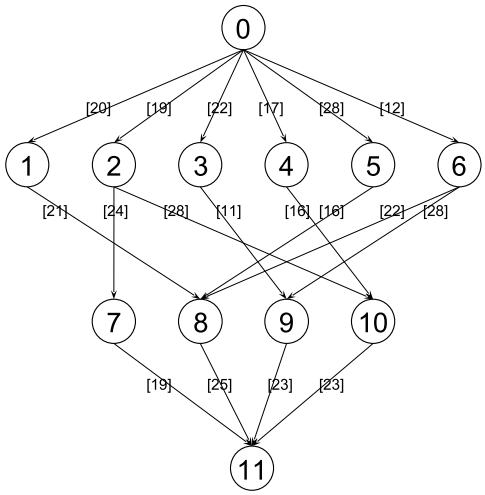
behind this threshold is to reduce the evaluation overhead in the selecting phase by dropping some schedules. Apparently, if a produced schedule $\Omega_{\mathcal{G}}$ does not have a reasonably short $\overline{M}_{\Omega}$, it is unlikely that the schedule will achieve a good average makespan, and so it is unnecessary to be recorded.

V. An Example

An example DAG with 12 tasks is used here for illustration purposes. Figure 3(a) shows the DAG structure and the size of data to transmit between two interdependent tasks. Three resources were assumed to run the DAG. For each task, the execution time is modelled as a random variable following gaussian (normal) distribution. Figure 3(b) gives values for the mean of each random task execution time (i.e., μ). In addition, the random variable is assumed to follow gaussian distribution in the interval $[0.5\mu,\ 1.5\mu]$, and the standard deviation is 0.167μ; values outside this interval are not considered. Figure 3(c) provides the data transmission rates between 3 resources. To implement MCS (as described in Figure 2), the loop in the producing phase is repeated up to 50000 times, the loop in the selecting phase is repeated 200 times, $\Delta = 0.02$, and HEFT [3] is employed as the DAG scheduling heuristic, $\mathcal{H}$.

Figure 4(a) depicts the schedule generated by MCS and the makespan this schedule can obtain using as actual task execution times the mean of each random task execution time. In contrast, the schedule generated by HEFT using the mean of task execution time, i.e., the mean static schedule defined in Eq.(12), is shown in Figure 4(b). It is apparent that MCS results in a significantly better makespan than HEFT for such a small DAG. The improvement is around 10% (the makespan of the schedule obtained with MCS is about 163.5, in contrast to the schedule obtained with HEFT which has a makespan of about 181.5). More importantly, the extra time needed to run MCS was less than 0.2 seconds, which may clearly be worth the effort.

It should be noted that, due to the iterative feature of MCS, the number of repetitions executed in both the selecting and producing phases is closely related to the quality of the makespan obtained by MCS, as well as the extra cost introduced. As mentioned before, 50000 repetitions in the producing phase and 200 repetitions (iterations) in the selecting phase have been used to obtain the schedule shown in Figure 4(a). The question is 'are that many repetitions necessary?'. To answer this question we need to understand the rate of convergence to good solutions, in other words, how quickly (after some repetitions) a good schedule is found. In order to investigate these issues, the performance of MCS in the example DAG was observed by varying the number of repetitions taken by MCS. We examined the results for up to 50000 repetitions in the producing phase and for 10, 50, 200 repetitions in the selecting phase.



(a) topology of DAG G

Task	R_0	R_1	R_2	Task	R_0	R_1	R_2
0	9.0	48.0	19.0	6	9.0	19.0	23.0
1	45.0	60.0	14.0	7	8.0	12.0	5.0
2	11.0	16.0	9.0	8	25.0	84.0	67.0
3	14.0	37.0	29.0	9	3.0	8.0	1.0
4	8.0	14.0	17.0	10	14.0	19.0	18.0
5	40.0	75.0	66.0	11	53.0	86.0	20.0

(b) mean of task runtime predictions of G on three heterogeneous resources

Connected Resources	Transmission time for per unit of data
R_0 and R_1	1.34
R_0 and R_2	1.72
R_1 and R_2	1.16

(c) transmission rates between resources

Figure 3: An example DAG with 12 nodes

Figure 5 shows the results of our investigation when applying MCS to the test random DAG. Figure 5(a) depicts how the number of produced schedules (N_{PS}) and the elapsed time evolve as the number of repetitions in the producing phase (R_P) increases. Figure 5(b)-(d) shows how the average makespan ($\overline{M}_\Omega$, obtained over 10, 50 or 200 repetitions respectively in the selecting phase) of the selected schedule is improved as the number of produced schedules grows.

Figure 5 suggests that it is not necessary to take too many repetitions in the producing phase to obtain a schedule which has a reasonably short average makespan. It can easily be observed from Figure 5(b-d) that about 1000 repetitions in the producing phase result in a good makespan. This number can be produced reasonably quickly. The dotted lines in Figure 5(a) show the time needed for 1000 and 10000 repetitions in the producing phase. Moreover, Figure 5(b)-(d) suggests that different settings for the number of repetitions in the selecting phase have little impact on the average makespan obtained by MCS. This indicates that it can be feasible for MCS to substantially reduce its overhead without significantly losing its performance by setting a small limit on the number of repetitions in both the producing and selecting phases, and subsequently applying MCS.

VI. Experimental Evaluation

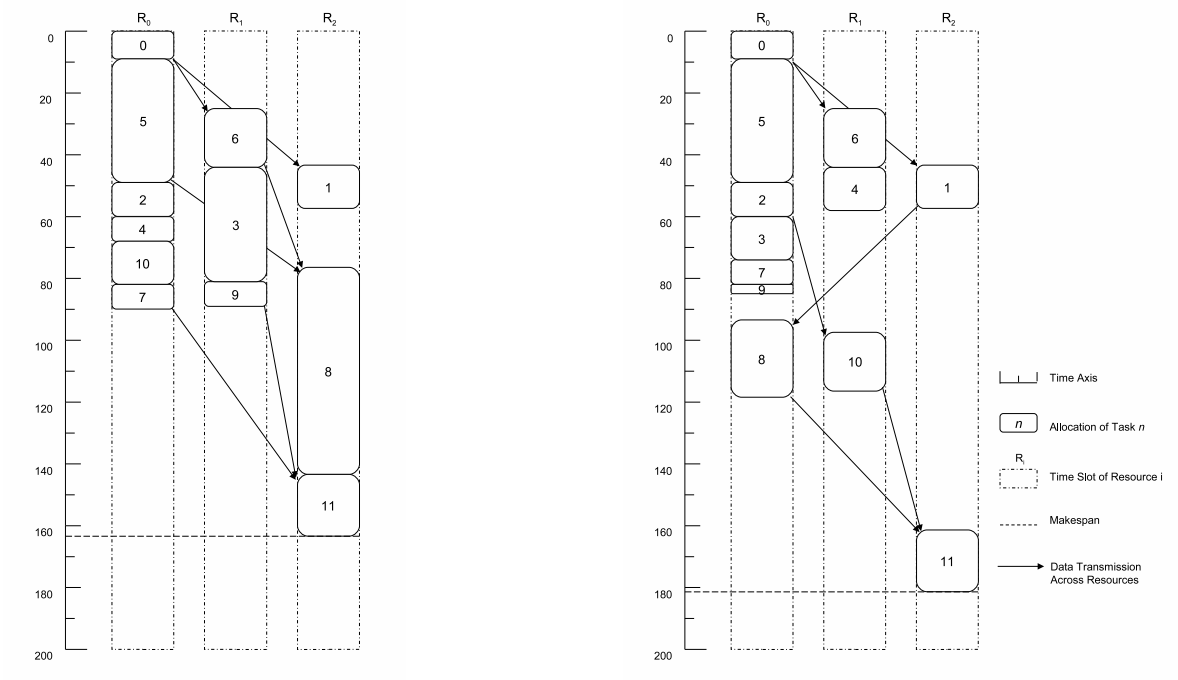
A. Settings

In order to evaluate MCS, our approach was incorporated into a DAG scheduling simulator that was used in other similar studies [12], [42], [43].

In order to assess the performance of MCS, several experiments were conducted to compare the makespan of the schedule obtained by MCS, employing HEFT [3] as the static scheduling heuristic $\mathcal{H}$, with the makespan of the schedules generated by another four approaches. These approaches are denoted by *Static*, *Autopsy*, *ReStatic* and *ReMCS* and are explained below.

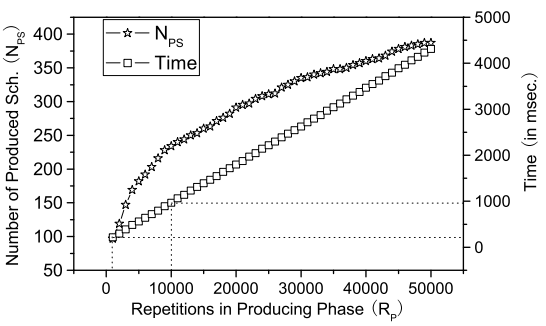
- *Static* refers to the schedule which is computed by applying $\mathcal{H}$ to the mean of the random task execution time predictions (i.e., $\mu_{i,p}$).
- *Autopsy* refers to the schedule which is constructed by $\mathcal{H}$ after the actual task execution times are known. This is not a realistic approach, as actual task execution times cannot be known when scheduling (only estimates), but it is included for comparison purposes.
- *ReStatic* (i.e., Static Schedule with Rescheduling) corresponds to the procedure which firstly generates an initial schedule by the Static approach and then recalls the heuristic used in the Static approach to reschedule all of the remaining non-executed tasks each time a task is about to begin execution.
- *ReMCS* (i.e., applying rescheduling to MCS) uses the result of MCS as the initial schedule, and then calls the heuristic used by MCS to reschedule all of the remaining non-executed tasks each time a task is about to begin execution. It should be noted that when making a rescheduling decision, ReMCS adopts the task execution time sample $\mathcal{P}_\mathcal{G}$, which is recorded with $\Omega_\mathcal{G}$, as input.

For the evaluation, we considered three DAGs, which are based on real-world workflow applications [1], [44]. The three DAGs are shown in Figure 6 and are: Montage [39] with 34 tasks, AIRSN [40] with 53 tasks, and LIGO [41] with 77 tasks. The method in [45] was adopted to model the heterogeneity of the mean task execution time estimate, which was randomly generated in the range of [1, 100]. In

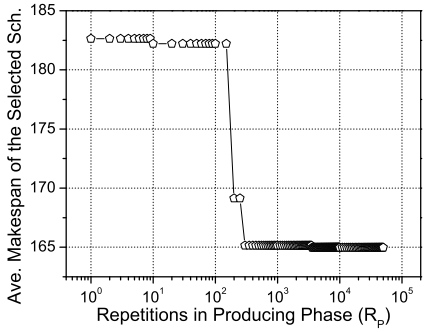


(a) MCS (b) HEFT

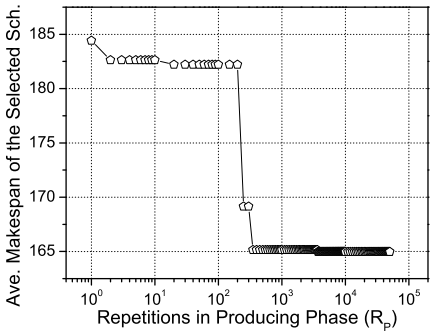
Figure 4: The schedule generated by MCS and HEFT, respectively



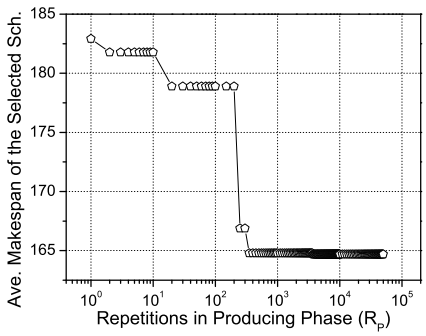
(a) N_{PS} and Time vs. R_P



(b) $\overline{M}_\Omega$ vs. R_P, 10 iterations in Selecting Phase

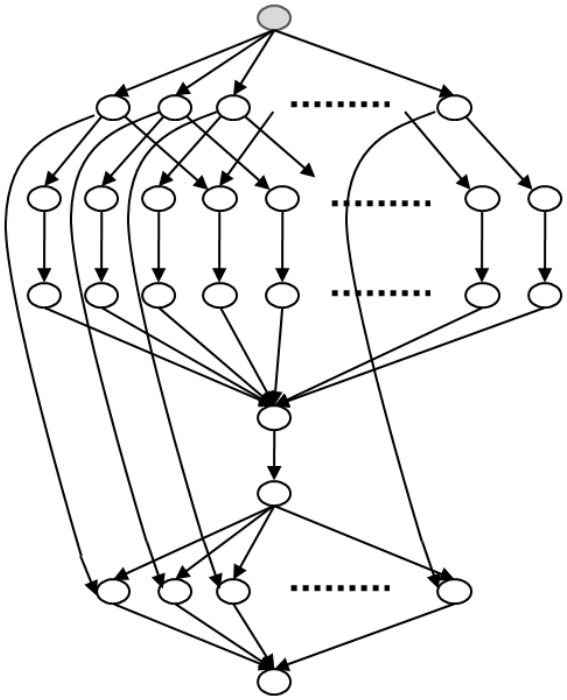


(c) $\overline{M}_\Omega$ vs. R_P, 50 iterations in Selecting Phase

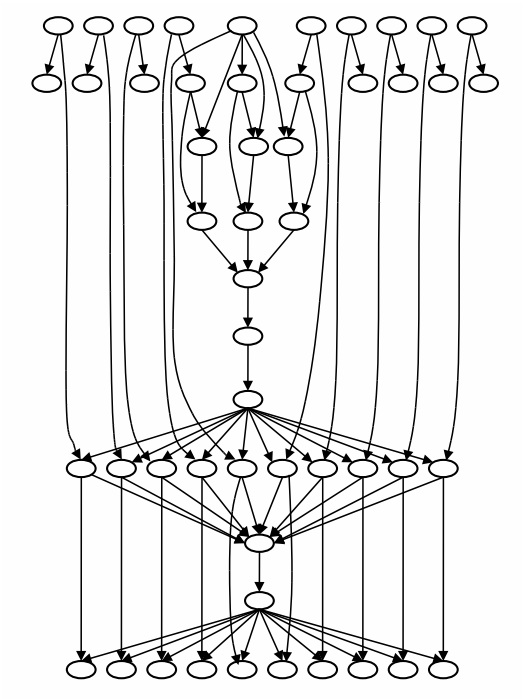


(d) $\overline{M}_\Omega$ vs. R_P, 200 iterations in Selecting Phase

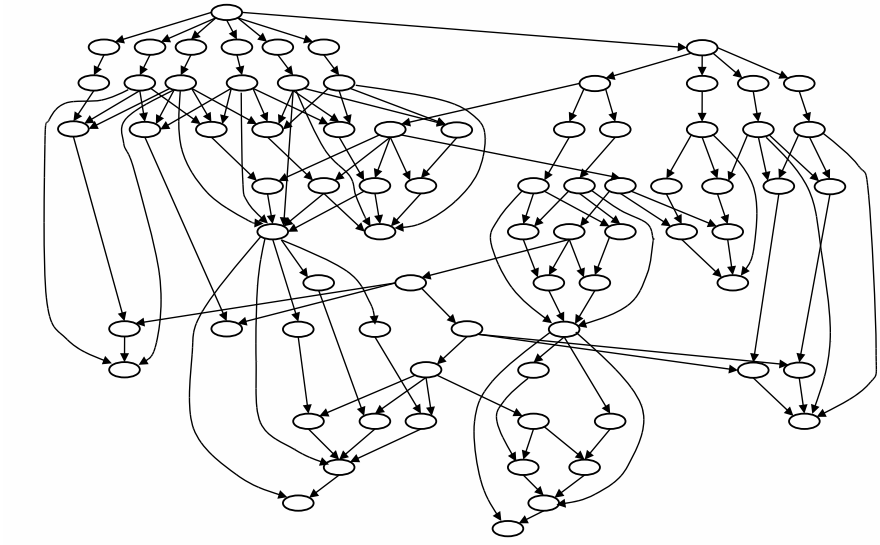
Figure 5: MCS performance with different number of repetitions (using the DAG in Figure 3)



(a) Montage [39]

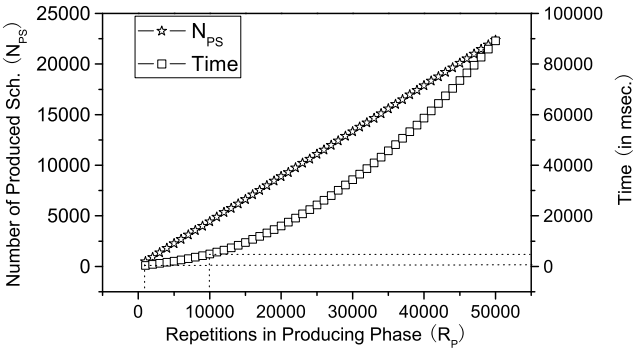


(b) AIRSN [40]

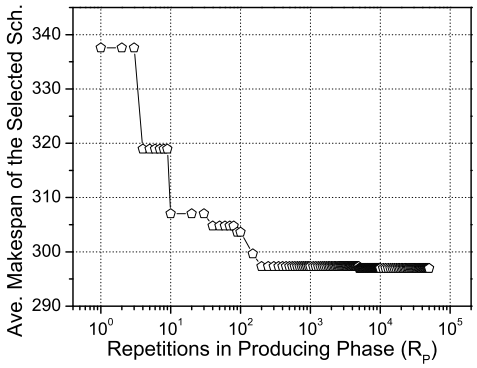


(c) LIGO [41]

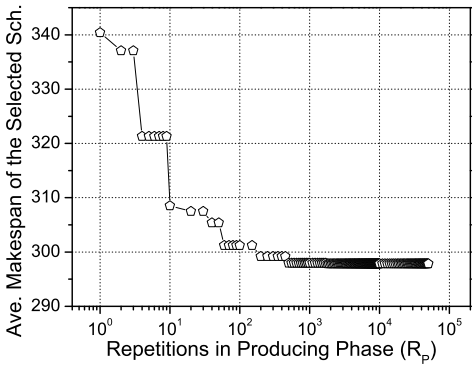
Figure 6: DAG applications used in the experiments



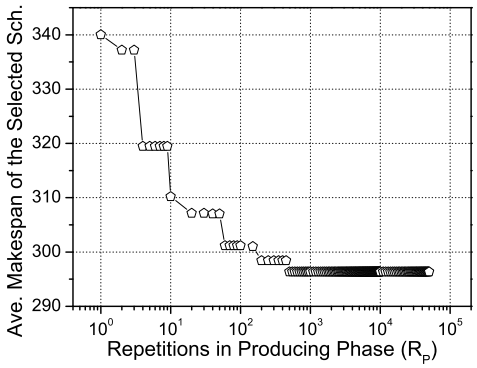
(a) N_{PS} and Time vs. R_P



(b) $\overline{M}_\Omega$ vs. R_P, 10 iterations in Selecting Phase



(c) $\overline{M}_\Omega$ vs. R_P, 50 iterations in Selecting Phase



(d) $\overline{M}_\Omega$ vs. R_P, 200 iterations in Selecting Phase

Figure 7: MCS performance with different number of repetitions, using a Montage DAG with 34 nodes

this method, in brief, two values are respectively selected from a uniform distribution at the range of [1, 10], and then the product of the two selected values is computed and adopted as a generation of a task execution time. The communication-to-computation ratio (CCR) was randomly chosen from the interval [0.1, 1]. It is assumed that the actual execution time of each task follows gaussian distribution within a certain boundary around the mean of execution time estimation (denoted by μ). Similar to the notion of QoI defined in [10], the notion of *Quality of Estimation* (QoE, denoted by $0 < \delta < 1$) was adopted to describe the upper boundary of percentage deviation which the actual task execution time may have with respect to the mean value (a value of zero means no deviation). The lower bound and the upper bound of task execution time are both set three standard deviations away from the mean. One standard deviation equals to $\mu * \delta/3$. In the random generation of a task execution time, the process repeats until a value within the range between the lower bound and the upper bound is generated.

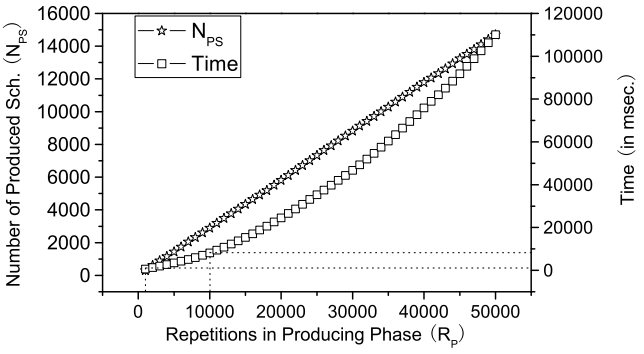
In the experiments, we considered five different values for QoE (0.2, 0.4, 0.6, 0.8, 1.0). For each of the four different scheduling approaches (mentioned above), which we used for comparison, the average makespan of the DAG was obtained over 100 samplings of the stochastic prediction for task execution times with the process of generating task execution times repeated another 50 times (hence, the makespan is averaged over 5000 values).

Finally, all the experiments were run on a PC with Pentium 4 CPU running at 3.2 Ghz and with 1 GB Memory.

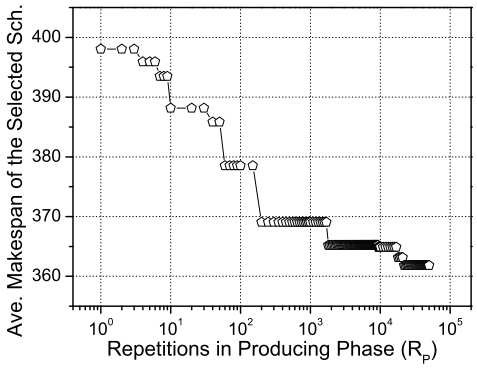
B. Specifying the number of repetitions

As observed in Section V, the number of repetitions required (in both the producing and the selecting phase) to obtain a makespan of good quality need not be that high. In order to determine this number, we carried out an exercise similar to the one performed in Section V with the example DAG, where we observed the average makespan with different number of iterations in the producing and selecting phases.

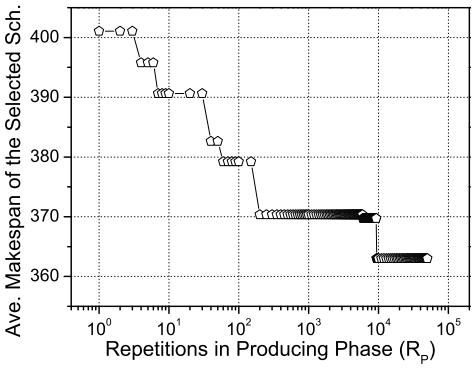
Once again, we considered up to 50000 iterations in the producing phase and 10, 50, 200 iterations in the selecting phase. For each of the three DAGs considered the results are given in Figure 7, Figure 8 and Figure 9, respectively.



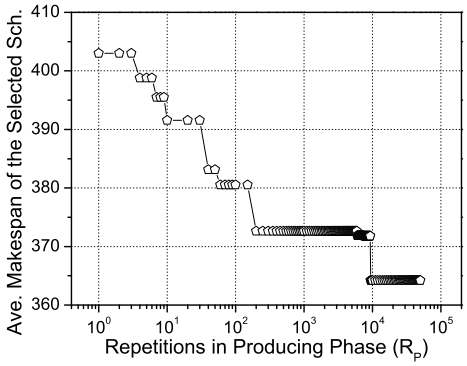
(a) N_{PS} and Time vs. R_P



(b) $\overline{M}_\Omega$ vs. R_P, 10 iterations in Selecting Phase



(c) $\overline{M}_\Omega$ vs. R_P, 50 iterations in Selecting Phase



(d) $\overline{M}_\Omega$ vs. R_P, 200 iterations in Selecting Phase

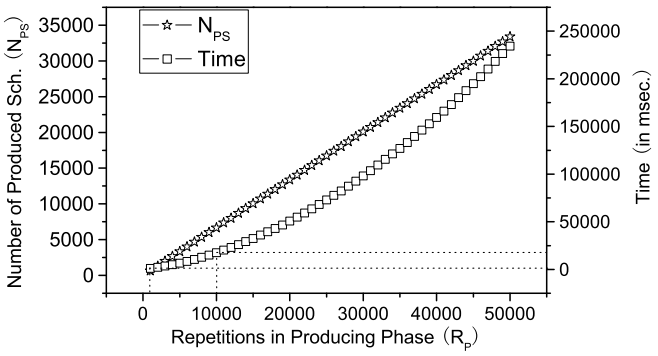
Figure 8: MCS performance with different settings of repetitions, using an AIRSN DAG with 53 nodes

A similar pattern as before can be observed. In all cases, it appears that 1000 repetitions in the producing phase are sufficient to obtain significant performance benefits. Also, the impact of the number of repetitions in the selecting phase appears to be small. Same as before the time taken for 1000 and 10000 repetitions in the producing phase is indicated by the dotted lines in Figure 7(a), Figure 8(a) and Figure 9(a).

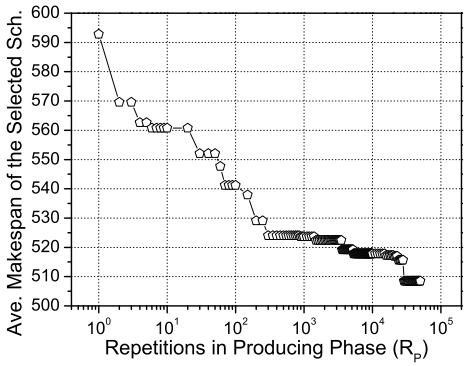
As a result, we determined empirically the number of repetitions in each phase. The loop of the producing phase (lines 3-7 of the algorithm in Figure 2) terminates when 50000 repetitions are completed or when the loop is executed for more than 30 seconds. Empirically, on the basis of the results in Figures 7 to 9, it seems that this terminating condition will allow us to produce a number of schedules that could result in good performance, yet without spending more than 30 seconds. For the loop in the selecting phase (lines 8-13 of the algorithm in Figure 2), we used 20 iterations. Finally, for the criterion of QUALIFIED (line 6 of the algorithm in Figure 2 and Equation 14) we used the value $\Delta = 0.02$.

C. Results

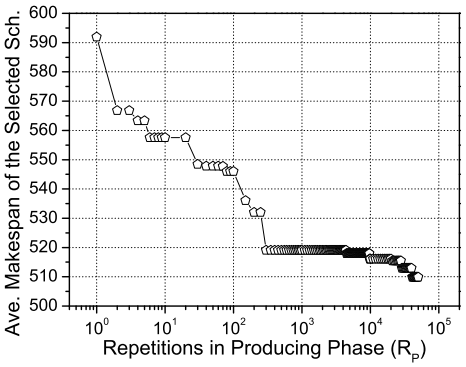
The results for each of the five different scheduling approaches and each of the three DAGs are shown in Figure 10. The results are normalized according to the performance of *Static*, which gives, consistently, the worst makespan. Recall that for each scheduling approach the results are averaged over 5000 runs (100 samplings repeated 50 times) As expected, MCS always results in a better average makespan than Static as run-time changes of task execution time occur; an improvement of 6% to 9% is usually achieved. It is also interesting to observe that almost always rescheduling the schedule produced by MCS (i.e., *ReMCS*) results in a better makespan than rescheduling the schedule generated by the static approach (i.e., *ReStatic*). This is an advantage of MCS, which generates a schedule by selecting (among many different schedules) the one with the best expected value for the makespan. By doing so, MCS shows good behaviour not only as a static DAG scheduling approach, but also as an initial scheduling approach when run-time rescheduling is performed. Of course, it should be mentioned here that rescheduling is unavoidable in a more dynamic context. For example, for values of QoE higher than 0.4, *ReMCS* always



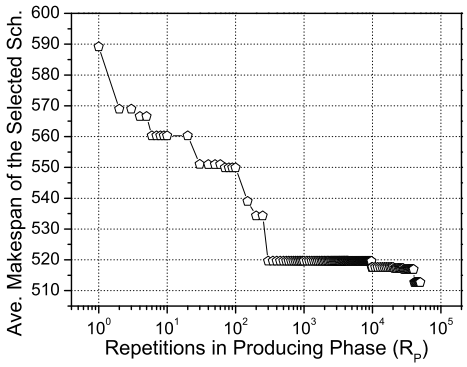
(a) N_{PS} and Time vs. R_P



(b) $\overline{M}_\Omega$ vs. R_P, 10 iterations in Selecting Phase



(c) $\overline{M}_\Omega$ vs. R_P, 50 iterations in Selecting Phase



(d) $\overline{M}_\Omega$ vs. R_P, 200 iterations in Selecting Phase

Figure 9: MCS performance with different settings of repetitions, using a LIGO DAG with 77 nodes

results in a better makespan than *MCS*. Compared to *Static*, the makespan improvement by *ReMCS* can be 15%-20% better when QoE equals to 0.8 or 1.0. Finally, it is interesting that, when the distribution of the random task execution time estimation is known, applying *MCS* and/or *ReMCS* (which, by construction, assess many different schedules before choosing one) can achieve better results than HEFT with perfect task execution time estimation, i.e., *Autopsy*.

Aside from the makespan obtained by MCS and its comparison with other scheduling approaches, we assessed the execution time of MCS for different DAGs and different values of QoE. The results are shown in Table I. We can observe that the execution time is always below 45 seconds, with the execution time of the producing phase taking most of the time.

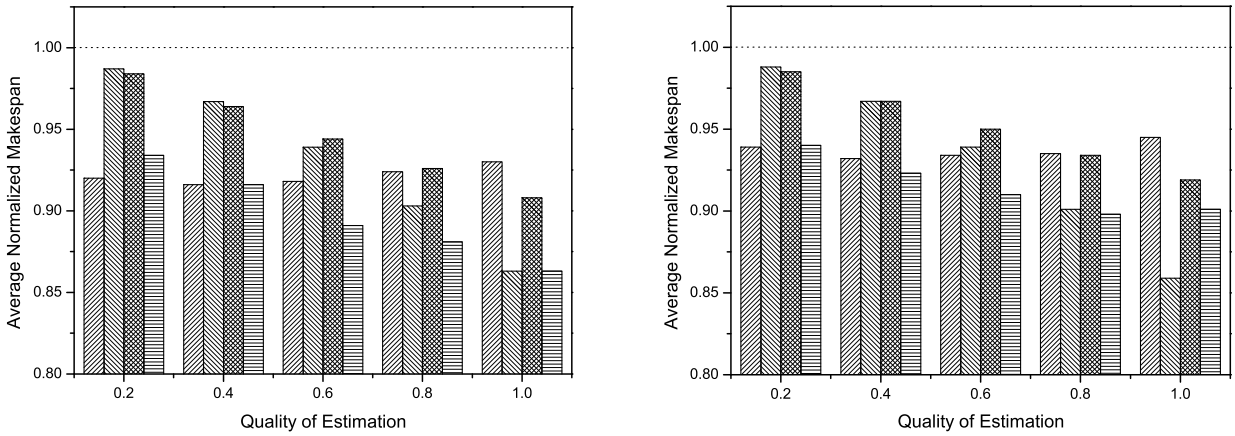
D. Summary of observations

The key findings from the aforementioned evaluation can be summarized as follows:

- It is possible to obtain a good full-ahead static schedule that performs well under prediction inaccuracy, without too much overhead. A good schedule can be found without many random samplings in producing phase, and selected by only a few random samplings in selecting phase.
- MCS, which has a more robust procedure for selecting an initial schedule, results in better performance when rescheduling is applied (as a result of run-time deviations). This means that the option of rescheduling *per se* is not sufficient to give good results when there are run-time deviations. A good initial (static) schedule is also important.
- Although it is recognized that the Monte Carlo approach can potentially add a significant overhead, by carefully crafting the termination conditions of the producing and selecting phases in MCS, it is possible to strike a good balance between the acceptable scheduling overhead and the desired performance improvement.

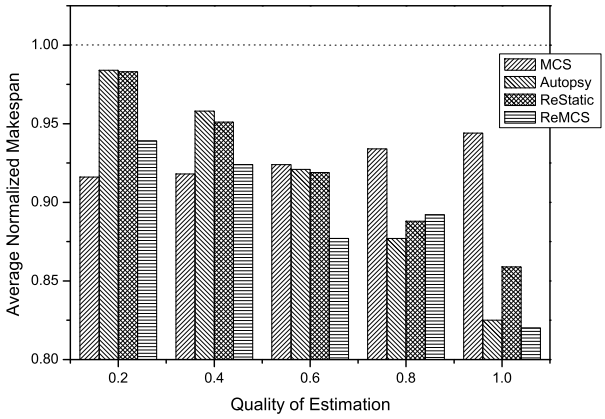
VII. Conclusion

This paper has proposed a full-ahead scheduling scheme to tackle DAG scheduling problems with stochastic performance prediction. Based on the Monte-Carlo method,



(a) Montage

(b) AIRSN



(c) LIGO

Figure 10: Average makespan of five different scheduling approaches normalized to static scheduling using HEFT

Table I: Analysis of the behaviour of MCS for the three DAGs and five QoE values considered. For each combination of QoE (δ) and DAG the three values mentioned indicate: the execution time of the whole MCS (in seconds), the execution time of the producing phase alone (in seconds), and the average number of produced schedules (in parentheses)

δ (QoE)	Montage	AIRSN	LIGO
0.2	37.01/29.04(11973)	43.67/30.00(11113)	44.61/30.00(9901)
0.4	37.98/29.38(12790)	40.33/30.00(8392)	40.70/30.00(7276)
0.6	35.31/28.58(9848)	35.89/29.66(5192)	36.32/30.00(4292)
0.8	30.73/26.06(6736)	32.10/28.91(2692)	32.99/30.00(2033)
1.0	24.91/22.08(3979)	29.09/27.80(1112)	31.08/30.00(742)

the proposed approach, MCS, seeks such a schedule which can obtain a competitive average makespan over the varying stochastic task execution times. Extensive evaluation using simulation was undertaken; the experimental results demonstrated that MCS built on the top of a deterministic scheduling heuristic, such as HEFT, always outperforms the results obtained by applying the heuristic directly to the DAG. In some cases, the schedules obtained by the most sophisticated procedure used by MCS, with or without rescheduling, could even perform better, on average, than those (static) schedules built after the accurate execution time prediction for each task on each resource is known.

This work still leaves space for further improvement. First, different probability distributions for modelling task execution time can be tried. Second, the stochastic approach can be used to model the uncertainty of communication time. Third, the Monte Carlo approach could be assessed with more different types of DAGs both in terms of size and in terms of structure. Fourth, further analysis could try to determine more useful terminating conditions for the producing phase and the selecting phase and/or threshold values that optimize the behaviour of the algorithm. Addressing these issues may allow users to tune MCS in a way that takes into account both the time they are prepared to afford to find a good schedule and the expected performance benefits under different circumstances.

REFERENCES

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 5, no. 25, pp. 528–540, 2009.

[2] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[3] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 13, pp. 260–274, Mar. 2002.

[4] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 13th Heterogeneous Computing Workshop*, 2004, pp. 111–124.

[5] A. Radulescu and A. J. C. van Gemund, "Fast and effective task scheduling in heterogeneous systems," in *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*, 2000, pp. 229–238.

[6] A. Lastovetsky and J. Twamley, "Towards a realistic performance model for networks of heterogeneous computers," in *High Performance Computational Science and Engineering*, ser. IFIP International Federation for Information Processing, M. Ng, A. Doncescu, L. Yang, and T. Leng, Eds. Springer Boston, 2005, vol. 172, pp. 39–57.

[7] G. Malewicz, I. Foster, and A. L. Rosenberg, "A tool for prioritizing DAGMan jobs and its evaluation," *Journal of Grid Computing*, vol. 5, no. 2, pp. 197–212, 2007.

[8] G. Cordasco, G. Malewicz, and A. L. Rosenberg, "Extending IC-scheduling via the sweep algorithm," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 201–211, 2010.

[9] T. Szepieniec and M. Bubak, "Investigation of the DAG eligible jobs maximization algorithm in a grid," in *9th IEEE/ACM International Conference on Grid Computing*, 2008, pp. 340–345.

[10] R. Sakellariou and H. Zhao, "A low-cost rescheduling policy for efficient mapping of workflows on grid systems," *Scientific Programming*, vol. 4, no. 12, pp. 253–262, Dec. 2004.

[11] M. Wieczorek, A. Hoheisel, and R. Prodan, "Taxonomies of the multi-criteria grid workflow scheduling problem," in *Grid Middleware and Services*. Springer US, 2008, pp. 237–264.

[12] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of DAG scheduling heuristics," in *Grid Computing: Achievements and Prospects*, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds. Springer, 2008, pp. 73–84.

[13] J. M. Schopf and F. Berman, "Stochastic scheduling," in *Proceedings of 1999 ACM/IEEE Conference on Supercomputing*, 1999, pp. 48–68.

[14] L. Yang, J. M. Schopf, and I. Foster, "Conservative scheduling: using predicted variance to improve scheduling decisions in dynamic environments," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, 2003, pp. 31–47.

[15] A. Doğan and F. Özgüner, "Stochastic scheduling of a meta-task in heterogeneous distributed computing," in *International Conference on Parallel Processing Workshop*, 2001, pp. 369–374.

[16] ——, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, pp. 177–190, 2004.

[17] A. Kamthe and S.-Y. Lee, "A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 121b–121b.

[18] ——, "Stochastic approach to scheduling multiple divisible tasks on a heterogeneous distributed computing system," in *IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–11.

[19] L. Bittencourt, R. Sakellariou, and E. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *18th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2010, pp. 27–34.

[20] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 175–187, Feb. 1993.

[21] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *Proceedings of the 7th Heterogeneous Computing Workshop*, 1998, pp. 57–69.

[22] C. Boeres, G. Chochia, and P. Thanisch, "On the scope of applicability of the ETF algorithm," in *Parallel Algorithms for Irregularly Structured Problems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, vol. 980, pp. 159–164.

[23] H. Oh and S. Ha, "A static heuristic for heterogeneous processors," in *Proceedings of European Conference On Parallel Processing (Euro-Par'96)*, vol. 2, 1996, pp. 573–377.

[24] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid*, vol. 2, 2005, pp. 759–767.

[25] G. Q. Liu, K. L. Poh, and M. Xie, "Iterative list scheduling for heterogeneous computing," *Journal of Parallel and Distributed Computing*, vol. 5, no. 65, pp. 654–665, May 2005.

[26] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, vol. 1, no. 47, pp. 8–22, Nov. 1997.

[27] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, vol. 6-7, no. 42, pp. 465–475, Dec. 1996.

[28] B. Cirou and E. Jeannot, "Triplet: A clustering scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing*, 2001, pp. 231–236.

[29] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, 2000, pp. 445–450.

[30] ——, "A scalable task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing*, 2000, pp. 383–390.

[31] A. Dŏgan and R. Özgüner, "LDBS: a duplication based scheduling algorithm for heterogeneous computing systems," in *Proceedings of the International Conference on Parallel Processing*, 2002, pp. 352–359.

[32] S. A. Jarvis, L. He, D. P. Spooner, and G. R. Nudd, "The impact of predictive inaccuracies on execution scheduling," *Performance Evaluation*, vol. 1-4, no. 60, pp. 127–139, 2005.

[33] V. Shestak, J. Smith, A. Maciejewski, and H. Siegel, "Stochastic robustness metric and its use for static resource allocation," *Journal of Parallel and Distributed Computing*, vol. 8, no. 68, pp. 1157–1173, Aug. 2008.

[34] M. M. Lòpez, E. Heymann, and M. A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems," in *The Fifth International Symposium on Parallel and Distributed Computing*, 2006, pp. 199–207.

[35] A. Mahjoub, J. Pecero Sánchez, and D. Trystram, "Scheduling with uncertainties on new computing platforms," *Computational Optimization and Applications*, vol. 48, pp. 369–398, 2011.

[36] L.-C. Canon and E. Jeannot, "Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 532–546, April 2010.

[37] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A stochastic scheduling algorithm for precedence constrained tasks on grid," *Future Generation Computer Systems*, vol. 8, no. 27, pp. 1083–1091, Oct. 2011.

[38] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*. London: Methuen, 1975.

[39] G. B. Berriman, J. C. Good, A. C. Laity, A. Bergou, J. Jacob, D. S. Katz, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, and R. Williams, "Montage: A grid enabled image mosaic service for the national virtual observatory," *Astronomical Society of the Pacific Conference Series*, vol. 314, pp. 593–596, 2004.

[40] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, "A notation and system for expressing and executing cleanly typed workflows on messy scientific data," *SIGMOD Record*, vol. 3, no. 34, 2005.

[41] D. Brown, P. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis," in *Workflows for e-Science: Scientific Workflows for Grids*, 2006, pp. 39–59.

[42] W. Zheng, "Explorations in grid workflow scheduling," Ph.D. dissertation, The University of Manchester, 2010.

[43] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control in market-oriented environments," in *Proceedings of the 8th International Workshop on the Economics and Business of Grids, Clouds, Systems and Services (GECON 2011)*, to appear.

[44] I. Taylor, E. Deelman, D. Gannon, and M. Shields (Eds.), *Workflows for e-Science*. Springer, 2007.

[45] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proceedings of the 9th Heterogeneous Computing Workshop*, 2000, pp. 185–199.

BIOGRAPHIES

Wei Zheng received his PhD degree in Computer Science from the University of Manchester in 2010. He received the BSc degree and the Master Degree in Computer Science from Tsinghua University, in 2002 and 2005 respectively. He is now an Assistant Professor at Xiamen University. His research interests include scheduling, performance modelling and distributed computing.

Rizos Sakellariou is a member of the academic staff in the School of Computer Science at the University of Manchester since 2000. He has carried out research, for more than 20 years, on various topics related to parallel and distributed computing, with a focus on scheduling. He has published more than 90 research papers.