A Priority-Based Scheduling Heuristic to Maximize Parallelism of Ready Tasks for DAG Applications

Wei Zheng*, Lu Tang*
*School of Information Science and Technology
Xiamen University, China
Email:zhengw@xmu.edu.cn, tanglu@stu.xmu.edu.cn

Rizos Sakellariou†
†School of Computer Science
The University of Manchester, United Kingdom
Email:rizos@cs.man.ac.uk

***Abstract*—In practical Cloud/Grid computing systems, DAG scheduling may be faced with challenges arising from severe uncertainty about the underlying platform. For instance, it could be hard to have explicit information about task execution time and/or the availability of resources; both may change dynamically, in difficult to predict ways. In such a setting, the development of various kinds of just-in-time scheduling schemes, which aim at maximizing the parallelism of ready tasks of DAG, seems to be a promising approach to cope with the lack of environment information and achieve efficient DAG execution. Although many attempts have been tried to develop such just-in-time scheduling heuristics, most of them are based on DAG decomposition, which results in complicated and suboptimal solutions for general DAGs. This paper presents a priority-based heuristic, which is not only easy to apply to arbitrary DAGs, but also exhibits comparable or better performance than the existing solutions.**

***Keywords*-DAG; scheduling; just-in-time; priority-based;**

I. Introduction

The popularity of Internet encourages distributed computing platforms such as Clouds [1] or Grids [2] to orchestrate a vast number of computing resources to execute massive computational applications, especially the group of workflow applications which are often derived from scientific problems in the fields of mathematics, astronomy, etc [3]. These applications normally consist of tasks with complex dependencies, and can be represented by a Directed Acyclic Graph (DAG, for short).

Although many full-ahead static heuristics have been proposed for DAG scheduling to minimize the makespan (the whole application execution time) for heterogeneous systems [4], [5], [6], [7], [8], these approaches may not be suitable for highly dynamic Grid/Cloud environments due to temporal unpredictability [9]: (i) the interdependent task communication over the Internet may not be stable; (ii) the non-dedicated resources in charge of executing the DAG tasks may exhibit uncertain availability and/or task processing rate. In contrast, a just-in-time scheduling scheme attempts to minimize the impact caused by temporal uncertainties by making scheduling decisions on-the-fly [10], which means that the allocation of a task will not be decided until the task becomes ready for execution. With increasing interest in the topic in recent years, a just-in-time scheduling scheme has been adopted in practical projects such as Condor [11], where simple scheduling strategies, for instance FIFO, have been used. In large scale computation environments, from a user's point of view, free resources may quickly become busy if not allocated immediately [12]. Therefore, a scheduling pattern which allocates ready tasks to resources immediately when they become available, is encouraged. In this case, it has been observed that, due to the aforementioned temporal unpredictability, using FIFO to sequence the allocation of tasks may lead to an ineffective execution of an application with complex task dependencies [13]. This is because FIFO, which does not have any deliberate prioritization of task allocation, may encounter a so-called gridlock [13], namely, there are no ready tasks for allocation when a set of resource requests arrive.

In order to minimize the risk of encountering this situation, a new scheduling goal called IC-scheduling has been suggested by a series of papers [13], [9] to schedule the tasks of a DAG application in such an order that the number of ready tasks generated during execution is maximized for assignment to the resources becoming available. Intuitively, once the tasks of an application are executed in such an order, the application should obtain a considerable parallel speedup and the resource requests should be better utilized no matter how the dynamic resources behave. Following the suggested scheduling goal, a decomposition-based heuristic named IC-Optimal (ICO, hereafter) and its extension have been proposed in [9]. It has been shown in [14] that ICO outperforms some simple scheduling strategies maximizing the number of ready tasks, and minimizing the makespan. However, it has been recognized that many DAGs do not admit schedules that are optimal under IC-scheduling [15]. This observation motivated the development of another new paradigm called $AREA$-Oriented scheduling (AO-scheduling, for short) [16], which weakens the often-unachievable demand that the number of ready tasks is maximized at every step to the always-achievable demand that this number is maximized on average.

The strategy of maximizing the number of ready tasks may be promising for DAG scheduling in dynamic computing environments. However, both ICO and AO-scheduling are restricted by their decomposition-based design feature, and therefore have drawbacks of being highly complicated and overlooking global optimization. This indicates the necessity and the possibility of developing a new approach. Given this motivation, this paper proposes a novel priority-based heuristic (PB), which is easy to apply to an arbitrary DAG topology and performs better than ICO and AO-scheduling in executing DAG applications. The proposed heuristic is expected to provide an efficient and effective solution of just-in-time scheduling scheme for dynamic Grid/Cloud computing systems.

The remainder of the paper is organized as follows: the background information and the problem statement are presented in Section II; the related work is reviewed in Section III; the proposed PB heuristic is introduced in Section IV with a case study; the performance of the heuristic is evaluated in Section V; finally, a summary is provided in Section VI.

II. Background

In this section, we describe the DAG application model, the execution model, the quality metrics used in our work, and then state the problem we are going to address.

A. DAG Model

We focus on workflow applications which can be represented by a DAG modelled as follows: a DAG $G = \{N, E\}$ is a directed graph consisting of a set of nodes N and a set of edges E, each of which is of the form $(i \to j)$, where $i, j \in N$. A node i represents a workflow task, and an edge $i \to j$ denotes the inter-task dependency between i and j. The execution of j cannot begin until the execution of i has been completed, and j becomes a *ready node* when all of its parents are 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*, and childless nodes *sink nodes*. For standardisation, it is specified that the DAG has a single entry node and a single exit node, since all DAGs with multiple entry or exit nodes can be equivalently transformed to this specification. Apparently, an entry node of G must be a source node, and an exit node must be a sink node.

B. Execution Model

Similar to the description in [14], it is assumed that a DAG application is executed in a *batch mode* (a variant of which is studied in [17]): A READY pool is maintained to hold the ready tasks for the assignment to task execution requests from resources. These available resources appear batch by batch. As each batch arrives, the ready tasks are allocated by means of one task per resource. If more resources arrive than ready tasks, the unused resources will simply disappear, and can be regarded as utilized by other applications. If fewer resources arrive than ready tasks, the unallocated tasks will be returned to the pool. Ideally, it can be assumed that all of the tasks allocated at the ith batch will be completed before the $(i+1)$th batch arrives. More concretely, given that a DAG with n nodes is being executed with a schedule S in batch mode, when the ith coming batch appears with r_i resources and there are t_i ready tasks in the ready-task pool, then $\min(r_i, t_i)$ tasks will be allocated and executed. Suppose that resource batches arrive constantly until all of the tasks are completed, the duration between the moment when the first batch arrived and when the last task completed is recorded as the metric *batched-makespan*.

C. Quality Metrics

Based on the batch mode execution, our ultimate goal is to determine a schedule S for the given DAG G (a permutation of tasks indicating the order of assigning tasks to resources) so that the batch-makespan is minimized.

When a batch of resources arrives, the goal is to maximize the number of ready tasks for mapping to new resources. Suppose that tasks will be executed and completed in the order of their allocation [9], the goal becomes to produce as many ready tasks as possible after each task execution. Let $NR_S(i)$ denote the number of ready tasks appearing at the completion of the ith task in the order of execution. Obviously, $NR_S(n-1) \equiv 1$ due to the DAG standardisation adopted. A schedule S^* is called the *optimal schedule* if it maximizes the number of ready tasks at the completion of each task, i.e.,

$$(\forall i) NR_{S^*}(i) = \max_{S' \in S_G} \{NR_{S'}(i)\} \tag{1}$$

where S_G is the set of all possible schedules of G and $0 \leq i < n$.

As many DAGs do not admit any optimal schedule [13], for these DAGs, a metric called $AREA$ (denoted by $V(S)$) as defined below, is proposed in [15] in order to describe a sort of implicit quality of a schedule.

$$V(S) = \sum_{i=0}^{n-1} NR_S(i) \tag{2}$$

Apparently, unlike the IC-optimal schedule which may not exist for many DAGs, a schedule that maximizes $AREA$ exists for any DAG.

We provide the following simple example for illustrative purposes. Given a DAG G with 8 nodes as depicted in Fig. 1, where a number i inside a node represents Task i (T_i for short), Table 1 shows the scheduling steps when G is scheduled in order $\{T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$. It can be easily computed that the $AREA$ is 12.

D. Problem Statement

Based on these models, the scheduling problem we are going to address is to sort all of the tasks of a given DAG into a suitable order to produce as many ready tasks as possible

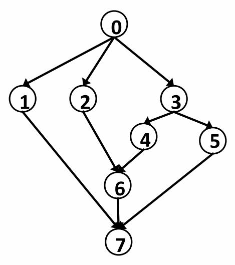

Figure 1. A DAG example with 8 nodes

	The Task Scheduled and Completed	Current Number of Ready Tasks
scheduling step 0	T_0	3
scheduling step 1	T_1	2
scheduling step 2	T_2	1
scheduling step 3	T_3	2
scheduling step 4	T_4	2
scheduling step 5	T_5	1
scheduling step 6	T_6	1
scheduling step 7	T_7	0

Table I
DETAILS OF SCHEDULING STEPS FOR SCHEDULE $\{T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$

while the batch execution continues, so as to minimize the batch-makespan.

III. RELATED WORK

The DAG execution model presented in Section II can be formalized by the Internet-Computing (IC) Pebble Games which uses pebbles to model the execution of a DAG. The placement and/or removal of various types of pebbles are used to represent the transition of task status (for example, ready and completed). Such games have been studied in [18], [19], [17], [20] for executing DAGs on the Internet. These studies were extended to the ICO algorithm proposed in [9] to obtain optimal schedules for DAGs with some specific structures. Simulation experiments carried out in [14] indicate that ICO significantly improves the execution time of a large class of DAGs over three simple, intuitively compelling scheduling heuristics. Malewicz et al. [9] extended the ICO algorithm to a practical heuristic applied in the Condor Project [11], and the usefulness of its implementation was assessed in [12], [21].

In order to improve the applicability of ICO, a series of papers [22], [23], [24], [25], [26], [27] have been published. However, there are still many DAGs that do not admit an IC-Optimal schedule. This fact spawned the development of $AREA$-oriented Scheduling (AO-scheduling). The concept of $AREA$-Maximization is first proposed in [15], and then extended by a series of works [28], [16], [29], [30], [31]. Work presented in [28], [29], [30] focuses only on Series-Parallel DAGs (SP-DAG, for short), a specific topology of DAG for which the $AREA$-Optimization has been proved to be achievable in polynomial time. In contrast, our work is not limited to any specific DAG. In [31], the $AREA$-MAX scheduling problem is proved to be NP-complete and two heuristics, one based on Sidney decomposition and the other based on Linear Programming, were proposed for solving the problem. However, $AREA$ is the only metric considered in [31]. In our work, we evaluate heuristics in terms of not only $AREA$ but also batched-makespan. It is worth mentioning that, for a DAG application, batched-makespan is the ultimate performance metric while $AREA$ is not. Moreover, as we will show latter, a better result of $AREA$ does not always lead to a better batched-makespan.

We consider the work presented in [16] as the closest related work, where the AO-scheduling heuristic (AREA-Oriented, named AO, hereafter) is proposed. The AO heuristic is applicable for general DAGs and has been evaluated by $AREA$ and makespan in [16]. We implement the AO heuristic described in [16] and use it as the competitor to our heuristic in the evaluation.

IV. THE PRIORITY-BASED (PB) HEURISTIC

In this section, we firstly describe the details of our proposed heuristic, PB, and then provide a case study in which PB has a better makespan than AO but worse $AREA$. We try to explain the reason behind this result.

A. Proposed Heuristic

The key idea of the PB heuristic is to award each DAG node a numerical priority to describe its ability to produce ready tasks. A node, which can enable more ready tasks instantly or potentially (expressed by the priority) by its completion, is considered stronger and should be executed preferentially. This principle determines the prioritized sequence of task execution, i.e., the result schedule. In contrast to the ICO/AO heuristic, which prioritizes DAG nodes in a decomposition manner, the PB heuristic calculates the priority of nodes according to their inter-dependencies, for example, in-degrees (the number of parent tasks). This approach has the following advantages:

- The risk of overlooking the optimal schedule can be reduced by considering the schedule globally instead of locally as a decomposition-based approach (such as ICO and AO) does;
- The heuristic can be applied to DAGs with any topological structure;
- The heuristic is easy to implement as it manipulates simple numerical values instead of complex topological structures.

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

 1:  Compute the initial $DQ$, $LQ$, $EQ$ and $IQ$ for each node in $G$.
 2:  Add the entry node into the Ready List $L$.
 3:  while $L$ is not empty repeat
 4:      Schedule the node $v$ in $L$ with the highest priority $P$, the
           comparison of task priority is jointly decided by $DQ$, $LQ$
           and $EQ$ following the decision tree shown in Figure 3.
 5:      Remove $v$ from $L$.
 6:      Remove $v$ from $G$.
 7:      for each child $x$ of $v$ in $G$ do
 8:          Decrease the in-degree of $x$.
 9:          UpdatePriority($x$).
10:      endfor
11:      Add new ready tasks into $L$.
12:  endwhile

where UpdatePriority is a recursive procedure defined as follows:
UpdatePriority($currentNode$)
if the in-degree of $currentNode$ is NOT equal to 0
    Update the $IQ$ of $currentNode$.
    for each parent $p$ of $currentNode$ do
        Update the $EQ$ and $DQ$ of $p$.
        UpdatePriority($p$).
    endfor
endif
```

Figure 2. The PB Heuristic

Four concepts are defined to make up a priority to capture the capability of a node to produce the ready tasks. For each node:

Direct Quotient (**DQ**) depicts the direct contribution a node can make to producing ready tasks. This is defined as the number of tasks which become ready immediately after the completion of the current node. Apparently, to achieve an optimal schedule, every scheduled node must have the highest DQ.

Level Quotient (**LQ**) depicts a node's topological position in the DAG. This is defined as the maximum length from a node to the exit node. It is assumed that the exit node is at level 0. Then, a node with a maximum length l to the exit node is placed onto level l. Apparently, the entry node is located at the highest level. Given a collection of ready tasks, it is preferable that the node on the highest level is run first unless there is another node with higher DQ.

Export Quotient (**EQ**) **and Import Quotient** (**IQ**) are recursively defined. EQ is a value only used for two tasks which have the same DQ and LQ to distinguish the priority of tasks. IQ is not used to compare task priority, but to help calculate the value of EQ for each task. This can be illustrated by the following definition. Given a node v:

$$IQ_v = \begin{cases} 0 & : \ v \in S_{sole} \cup S_{ready} \\ (EQ_v + 1)/ID_v & : \ otherwise \end{cases} \quad (3)$$

$$EQ_v = \begin{cases} 0 & : \ v \in S_{sole} \\ \sum_{u \in Succ(v)} IQ_u & : \ otherwise \end{cases} \quad (4)$$

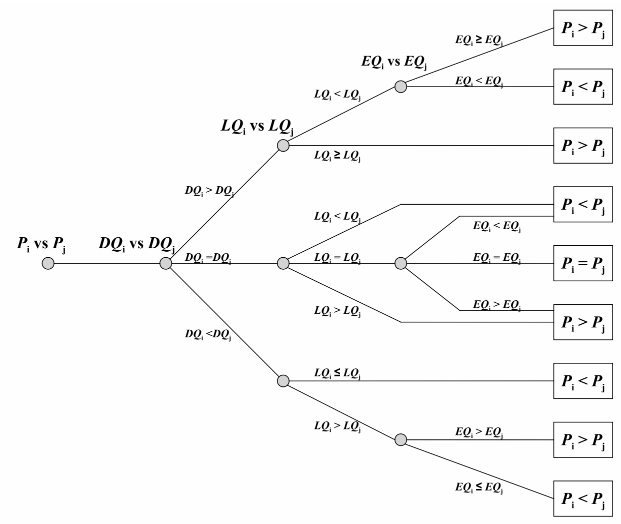

Figure 3. The decision tree of comparing task priority

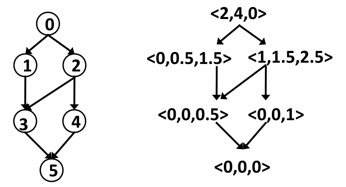

Figure 4. An example of computing the initial $< DQ, EQ, IQ >$

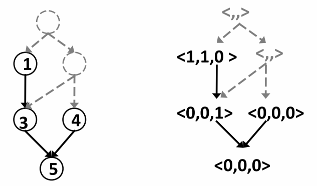

Figure 5. Updating $< DQ, EQ, IQ >$ after node 2 is scheduled

where ID_v means the in-degree of node v, $Succ(v)$ denotes the set of child tasks of v, S_{ready} is the set of ready nodes and S_{sole} is the set of *sole nodes*. A sole node is a node which can only be executed exclusively, for instance, the entry node and the exit node of a DAG. Therefore,

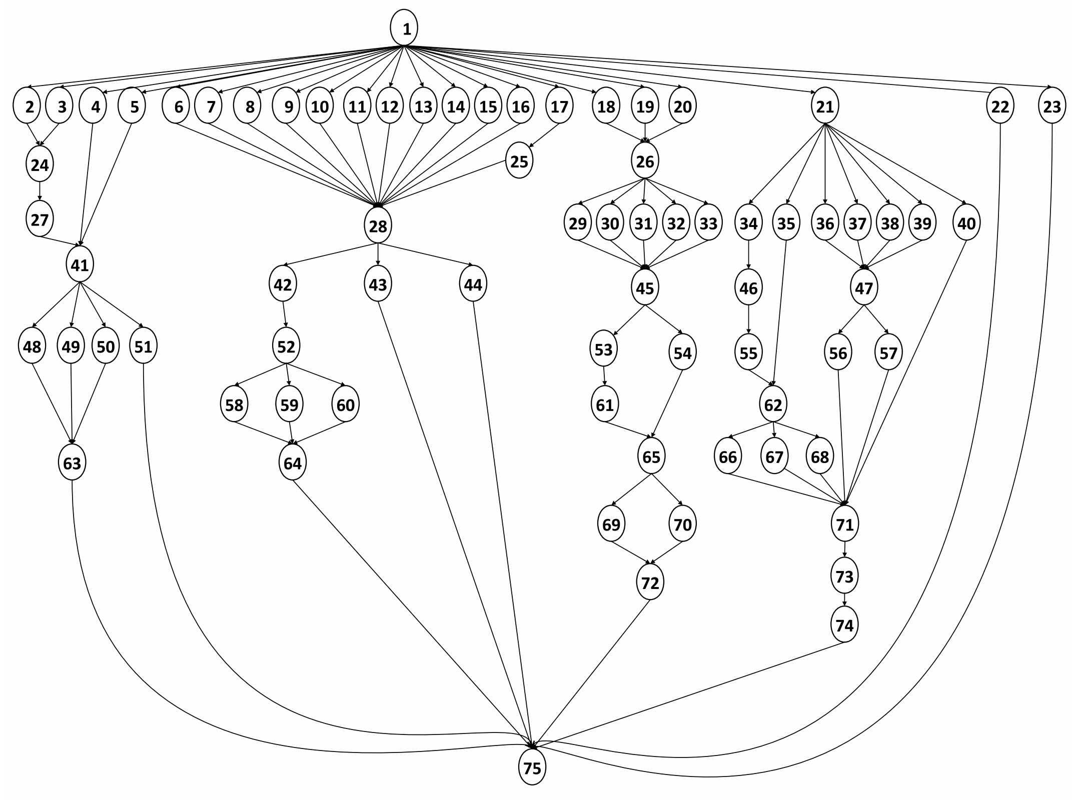

Figure 6. An SP-DAG example with 75 nodes

$EQ_{exit\_node} = 0$.

Based on the definitions of DQ, LQ, EQ and IQ, the pseudocode of the PB heuristic is presented in Figure 2. It should be noted that LQ does not change as the scheduling heuristic is executed, but DQ, EQ and IQ do. As an example, Figures 4 and 5 illustrate how DQ, EQ and IQ vary as the scheduling goes on.

Given the DAG G on the left-hand side of Figure 4, the initial values of DQ, EQ and IQ for each node can easily be calculated from their definition (results in the right half of Figure 4). Every pair of numbers in the position of the counterpart node is in the form of $< DQ, EQ, IQ >$. When node 0 has been completed, nodes 1 and 2 become ready. Therefore, the IQ value of nodes 1 and 2 turns to zero, while their DQ and EQ do not change. Subsequently, node 2 is selected because it has a higher DQ than node 1. When node 2 has been completed, DQ and EQ of node 1 are accordingly updated as shown in Figure 5.

B. Case Study

An example DAG with 75 tasks is used for illustration purposes. Figure 6 shows the DAG structure with all arcs pointing downwards. This DAG is an SP-DAG, for which the AO heuristic can guarantee an $AREA$-maximized schedule.

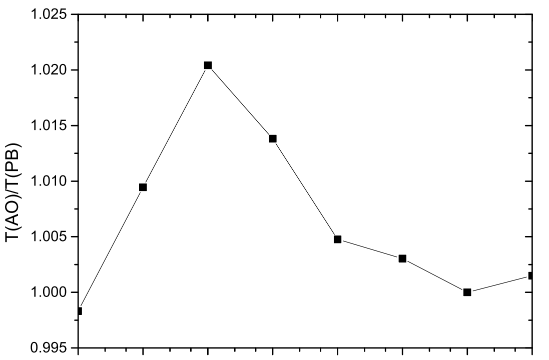

Figure 7. Makespan Comparison of PB and AO on the example DAG

For this DAG, the resulting schedule of AO is {1, 21, 18, 19, 20, 26, 34, 46, 55, 35, 62, 2, 3, 24, 27, 4, 5, 41, 17, 29, 30, 31, 32, 33, 45, 53, 61*, 54*, 65, 36, 37, 38, 39, 47, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 25, 28*, 42, 52, 69, 70, 66, 67, 68, 56, 57, 40, 71, 73, 48, 49, 50, 58, 59, 60, 63,

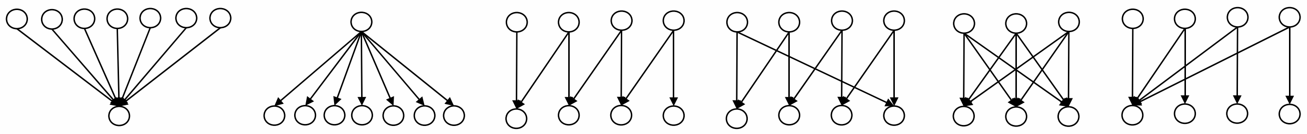

Figure 8. A sequence of six CBBBs (all arcs point downwards)

51, 64, 43, 44, 72, 74, 22, 23, 75} (here, '*' is used to mark the nodes we are going to discuss later) with an $AREA$ of 1490. In contrast, the resulting schedule of PB is {1, 21, 34, 20, 18, 19, 26, 46, 17, 32, 33, 30, 29, 31, 45, 53, 55, 35, 62, 2, 3, 24, 36, 39, 38, 37, 47, 12, 25, 14, 15, 16, 8, 11, 6, 10, 9, 13, 7, 28*, 42, 52, 5, 27, 4, 41, 61*, 54*, 65, 56, 40, 68, 57, 66, 67, 71, 73, 69, 70, 58, 59, 60, 48, 49, 50, 64, 72, 74, 51, 22, 44, 63, 43, 23, 75} with an $AREA$ of 1365, which is 125 down compared to AO.

To study the performance of AO and PB in the batch execution mode, we choose the number of resources arriving at each batch (i.e., batch size) from an exponential distribution with rate parameters λ = 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256, which means the mean of the batch size (denoted by μ) will be 2, 4, 8, 16, 32, 64, 128, 256 respectively. We assume that each allocated task can finish before the next batch comes. For each case of λ, we evaluate PB and AO fifty times and collect the batched-makespan on average. The result is shown in Figure 7, where $T(AO)/T(PB)$ means the time ratio between the average makespan of AO to the makespan of PB. In almost all cases of μ (the only exception is when $\mu = 2$), PB outperforms AO.

To understand the reason why PB obtains an averagely better makespan than AO, we specify the number of resources arriving at each batch (i.e., the batch size) and observe nodes that are allocated to resources at each batch. We tried to understand this by observing several times what happens if different patterns of batch size were used. We found that, in this specific example, a key difference that results in different batch-makespans of AO and PB is the executing sequence of node 61, 54 and 28, which are marked using a '*' in the provided scheduling details. The question is, when node 61, 54 and 28 are all ready for allocating to resources, which one should go first? The scheduling results indicate that AO will let 61 and 54 go before 28, while PB will do the opposite. Actually, one can easily see that node 28 is critical to the makespan. Completing node 28 at an earlier batch will usually result in a better batch-makespan. AO often executes node 28 one batch later than PB (because AO executes node 61 and 54 first), and consequently obtains a makespan worse than PB.

Essentially, in order to obtain an $AREA$-maximized schedule, the AO has to let node 28 appear late in its schedule. This is because node 28 is the node which has the most parents in the DAG and executing any of its parents contributes nothing to maximize the $AREA$. In contrast, PB is not driven by $AREA$-maximization. PB considers LQ (Level Quotient) and gives node 28 a good priority as this node has a relatively high LQ. As AO is kind of a decomposition-based approach, it is not easy to recognize the importance of nodes like node 28 in this example.

V. Experimental Evaluation

A. Settings

We compare three heuristics, PB, AO and FIFO in our experiments where the batch mode described in Section II is adopted. The FIFO scheduler organizes a given DAG's current ready tasks in a FIFO queue. When a batch of resources arrive, FIFO serves a free resource by dequeuing the task in the front of the queue. The nodes that are newly rendered ready for execution are enqueued in random order. We consider FIFO as a competitor to PB since FIFO is a basic scheduling approach also used by real systems such as Condor [11].

We randomly generate DAGs from two populations. One is SP-DAG, the other is CBBBC-DAG which is composed from a repertoire of Connected Bipartite Building Block DAGs (CBBBs, for short) [16]. As shown in Figure 8, the various structures of CBBB exemplify a variety of "real" computations, such as Fast Fourier Transform, accumulation tree, search tree etc. The way we used to randomly generate an n-*node* SP-DAG is the same as that used in [16]. We firstly specify the value of n and generate a random binary tree T. We then randomly designate each internal node of T either a series-composition node or a parallel-composition node with a random size m ($m \leq n$). We view T as the composition tree of an SP-DAG, and the designation of internal node proceeds until the size of T approaches n. The DAG shown in Figure 6 is an example of our random n-*node* SP-DAG. We randomly generate an n-*node* CBBBC-DAG also in a manner similar to that used in [16]. We firstly randomly choose an m-*node* ($m \leq n$) instance of one of the six structures of CBBBs shown in Figure 8, and use it as the current DAG. While the size of the current DAG is smaller than n, we continue choosing randomly an m-*node* CBBB instance (note that m is also random) and merge it into the current DAG. Figure 9 illustrates an instance of composing CBBBC-DAG. Note that in [16] CBBBC-DAGs are named $LEGO^{®}$-DAG. It is worth mentioning that, generating a

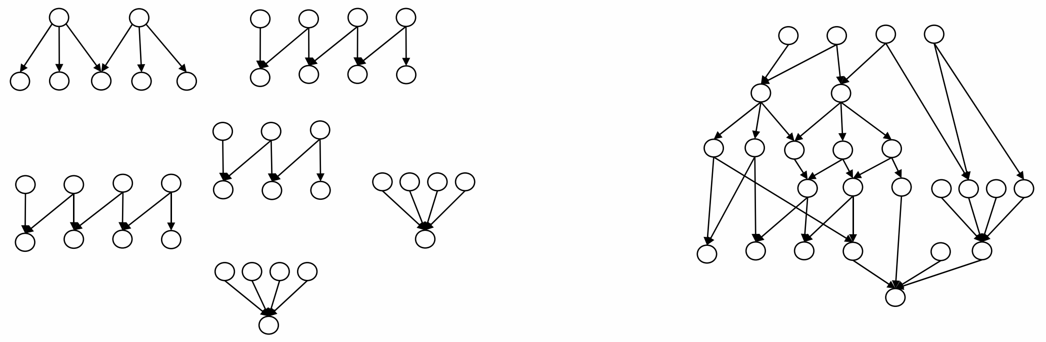

Figure 9. Composing six CBBBs into a CBBBC-DAG: (left) the CBBBs that compose the DAG; (right) the resulting CBBBC-DAG (all arcs point downwards)

random n-$node$ DAG means the generated DAG has roughly n nodes, because the random processes we use make it hard to specify the number of nodes exactly.

For each generated DAG, we craft three schedules by using PB, AO and FIFO, respectively. We compare the three schedules using two metrics. One is the batched-makespan and the other is the $AREA$. We obtained the batched-makespan using a probabilistic model of batch mode and the execution time of each allocated task. We choose the number of resources available at each batch from exponential distributions as specified in Section IV-B, which means our chosen value of batch size can be 2, 4, 8, 16, 32, 64, 128 and 256, respectively. We choose the intervals between neighbor batches from exponential distribution with rate parameter $\gamma = 1/10$, which means resource batches arrive every 10 time units on average. The execution time of an allocated task is randomly chosen from normal distributions. We have studied two distributions. The first distribution has a mean of 5 with standard deviation of 1 and the second distribution has a mean of 20 with standard deviation of 2. As the mean of batch arrival interval is 10, obviously, the former distribution means that an allocated task will be likely to finish before the next batch comes, while the latter means it will be unlikely.

B. Experimental Results and Discussion

1) $AREA$ Comparison: In this experiment, for each type of DAG we used, we generate random DAGs of sizes 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000. We run PB, AO and FIFO to each generated DAG and compute their $AREA$ results. Note that as FIFO has a random step during its scheduling, we repeat FIFO one hundred times and take the average $AREA$.

The $AREA$ comparison results for CBBBC-DAG and SP-DAG are shown in Figures 10 and 11, respectively.

The results for SP-DAG verify that AO can always provide an $AREA$-max schedule for SP-DAGs. For SP-DAGs, PB also significantly outperforms FIFO in the metric of $AREA$.

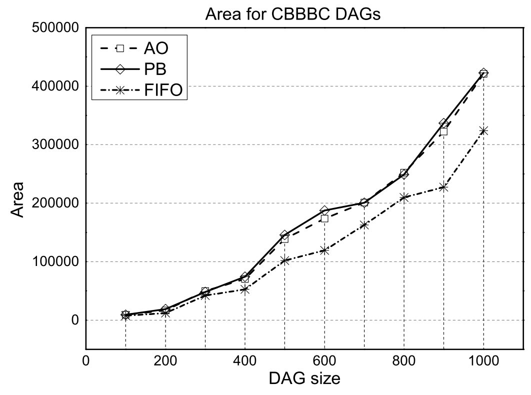

Figure 10. $AREA$ results for CBBBC-DAGs

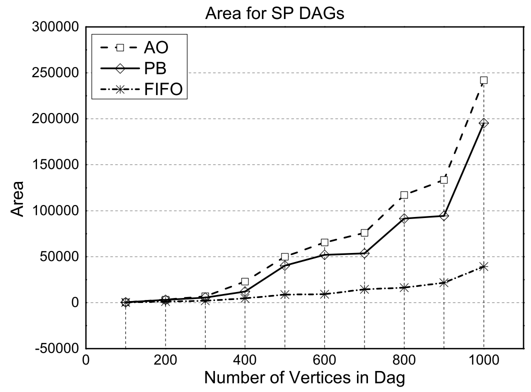

Figure 11. $AREA$ results for SP-DAGs

It can be observed that the amount of advantage in $AREA$ of AO and PB over FIFO rises as the DAG size increases.

For CBBBC-DAGs, PB and AO provide similar $AREA$ to each other in most of the DAG sizes considered. In some

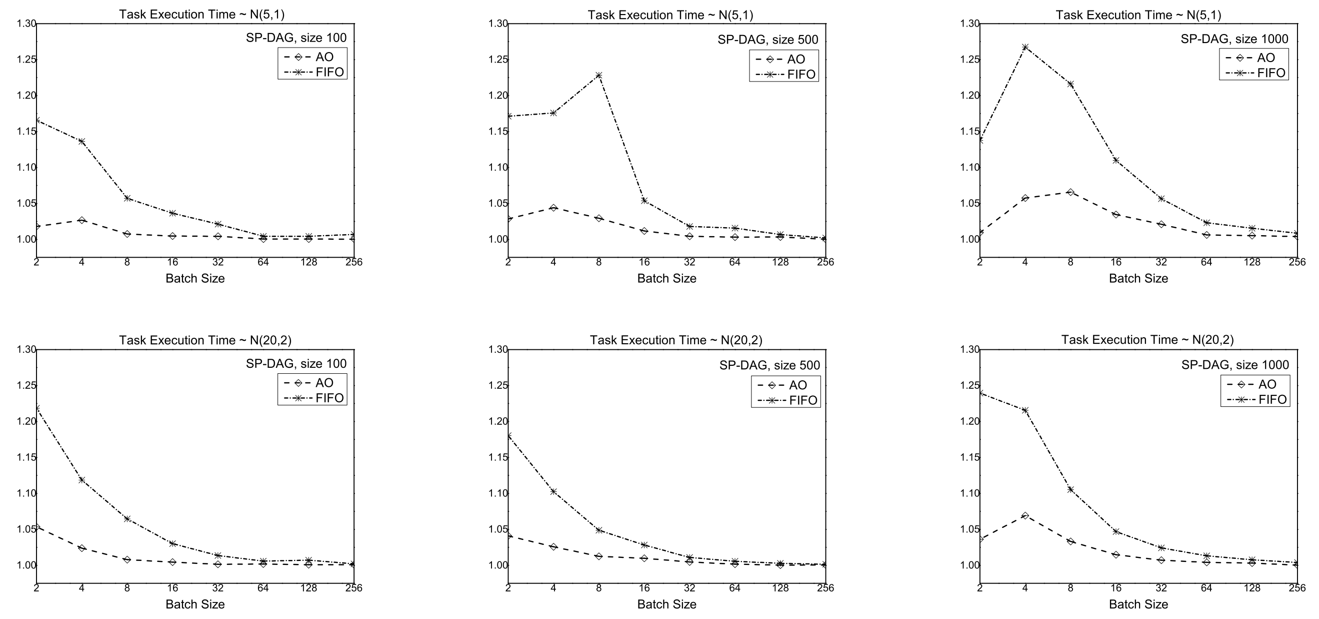

Figure 12. Ratio results for SP-DAGs

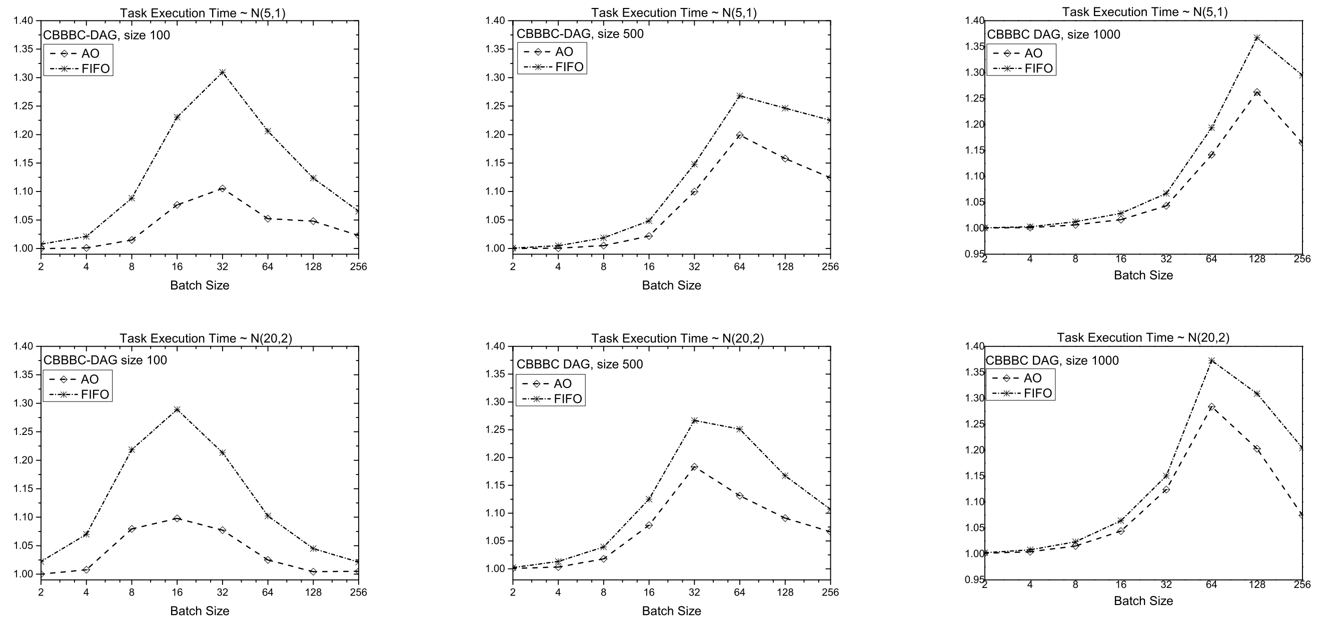

Figure 13. Ratio results for CBBBC-DAGs

of the cases, PB has an $AREA$ slightly better than AO. FIFO still performs the worst in $AREA$.

2) Makespan Comparison: In this experiment, for each type of DAG we used, we generate random DAGs of sizes 100, 500 and 1000, which stand for small, medium and large DAGs, respectively. We have considered 96 different experimental settings for each heuristic analyzed. Each setting is characterized by choosing from one of the two types of DAG considered, the three specified sizes of the DAG, the two distributions of task execution time, and the eight rates of the batch size. For each setting, we execute each DAG one hundred times and collect the averaged batched-makespan.

The performance of heuristic PB, as compared with its competitors, is illustrated via the ratios $T(AO) \div T(PB)$ and

$T(FIFO) \div T(PB)$, where $T(PB)$, $T(AO)$ and $T(FIFO)$ are the batched-makespans. Note that values of the ratio equal to or greater than 1.0 favor heuristic PB. The experimental results are shown in Figures 12 and 13, respectively. In each plot the X-axis indicates batch size, the Y-axis indicates the ratios for AO and FIFO.

Our first observation is that PB obtains a better average batched-makespan than AO and FIFO in all experimental settings. This result corroborates the observation from the case study presented in Section IV-B. Recall that in our AREA comparison experiments, AO always provides an $AREA$-max schedule for an SP-DAG. This result indicates that it is partially true that a schedule having a higher $AREA$ will have a better batched-makespan than another schedule. In the comparison of PB vs FIFO and AO vs FIFO, this statement is true, while for PB vs AO, it is not.

The observed advantage in the makespan of PB over AO and FIFO depends on four factors: the class of the DAGs used, the size of the DAG, the batch size, and the distribution of task execution time. First, PB has a more significant advantage over AO for CBBBC-DAGs than for SP-DAGs. This may correlate to the $AREA$s of the heuristics' schedules. Second, the advantage of PB increases as the size of the DAG grows. For the SP-DAG with 100 nodes and 500 nodes, PB shows an improvement to AO in the range of only $0 - 5\%$. For the SP-DAG with 1000 nodes, the improvement of PB over AO is up to 7%. This implies PB may perform significantly better for large-size DAG applications. Third, for a fixed-size DAG, there are always extremes of the batch size where the makespan will not depend on the scheduling heuristic. Between these extremes, there is a range of values of batch size where the scheduling heuristic has a strong influence on makespan. Finally, recall that some distributions of task execution time used in our experiments (when task execution time follows the distribution of $N(5, 1)$) may result in that an allocated task is likely to complete before the next batch of resources arrives, while other distributions (when task execution time follows the distribution of $N(20, 2)$) may result in the opposite. It can be observed that in the latter case, the advantage of PB over AO will be weakened.

VI. Conclusion

Just-in-time scheduling is a promising paradigm for scheduling complex applications with dependencies in highly dynamic distributed computing environments. This paper presents a novel just-in-time scheduling heuristic, PB, aiming at maximizing the parallelism of ready tasks during the execution of DAG application so as to minimize the makespan. Based on numerical priority, PB is easy to implement and is applicable to any DAG with arbitrary structure which indicates PB is compatible with any potential DAG applications. The simulated evaluation suggests that PB outperforms the existing just-in-time scheduling approaches under most experimental settings. This indicates that PB can become a competitive scheduling solution for the increasingly popular Internet Computing systems. Future work may evaluate the performance of PB in a wider variety of DAG topologies, and in real computing systems rather than simulation.

Acknowledgment

Partial support by the National Natural Science Foundation of China (Grant No. 61202361) is gratefully acknowledged.

References

[1] S. Yeo and H.-H. S. Lee, "Using Mathematical Modeling in Provisioning a Heterogeneous Cloud Computing Environment," *IEEE Computer*, vol. 44, no. 8, pp. 55–62, 2011.

[2] H. Casanova, F. Dufossé, Y. Robert, and F. Vivien, "Scheduling Parallel Iterative Applications on Volatile Resources," in *IPDPS*. IEEE, 2011, pp. 1012–1023.

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

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[5] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2004.1303065

[6] W. Zheng and R. Sakellariou, "Stochastic DAG scheduling using a Monte Carlo approach," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1673–1689, Dec. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0743731513001573

[7] 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, 2010.

[8] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics," *Grid Computing: Achievements and Prospects*, pp. 75–84, 2008. [Online]. Available: http://link.springer.com/content/pdf/10.1007/978-0-387-09457-1_7.pdf

[9] G. Malewicz, I. T. Foster, A. L. Rosenberg, and M. Wilde, "A Tool for Prioritizing DAGMan Jobs and its Evaluation," *J. Grid Comput.*, vol. 5, no. 2, pp. 197–212, 2007.

[10] M. A. Palis, J.-C. Liou, S. Rajasekaran, S. M. Shende, and D. S. L. Wei, "Online Scheduling of Dynamic Trees," *Parallel Processing Letters*, vol. 5, pp. 635–646, 1995.

[11] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[12] T. Szepieniec and M. Bubak, "Evaluation of Eligible Jobs Maximization Algorithm for DAG Scheduling in Grids," in *ICCS (1)*, ser. Lecture Notes in Computer Science, M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5101. Springer, 2008, pp. 254–263.

[13] G. Malewicz, A. L. Rosenberg, and M. Yurkewych, "Toward a Theory for Scheduling DAGs in Internet-Based Computing," *IEEE Trans. Computers*, vol. 55, no. 6, pp. 757–768, 2006.

[14] R. Hall, A. L. Rosenberg, and A. Venkataramani, "A Comparison of DAG-Scheduling Strategies for Internet-Based Computing," in *IPDPS*. IEEE, 2007, pp. 1–9.

[15] G. Cordasco and A. L. Rosenberg, "On Scheduling DAGs to Maximize Area," in *IPDPS*. IEEE, 2009, pp. 1–12.

[16] G. Cordasco, R. D. Chiara, and A. L. Rosenberg, "Assessing the Computational Benefits of AREA-Oriented DAG-Scheduling," in *Euro-Par (1)*, ser. Lecture Notes in Computer Science, E. Jeannot, R. Namyst, and J. Roman, Eds., vol. 6852. Springer, 2011, pp. 180–192.

[17] G. Malewicz and A. L. Rosenberg, "On Batch-Scheduling DAGs for Internet-Based Computing," in *Euro-Par*, ser. Lecture Notes in Computer Science, J. C. Cunha and P. D. Medeiros, Eds., vol. 3648. Springer, 2005, pp. 262–271.

[18] A. L. Rosenberg, "On Scheduling Mesh-Structured Computations for Internet-Based Computing," *IEEE Trans. Computers*, vol. 53, no. 9, pp. 1176–1186, 2004.

[19] A. L. Rosenberg and M. Yurkewych, "Guidelines for Scheduling Some Common Computation-DAGs for Internet-Based Computing," *IEEE Trans. Computers*, vol. 54, no. 4, pp. 428–438, 2005.

[20] G. Malewicz and A. L. Rosenberg, "A Pebble Game for Internet-Based Computing," in *Essays in Memory of Shimon Even*, ser. Lecture Notes in Computer Science, O. Goldreich, A. L. Rosenberg, and A. L. Selman, Eds., vol. 3895. Springer, 2006, pp. 291–312.

[21] T. Szepieniec and M. Bubak, "Investigation of the DAG Eligible Jobs Maximization Algorithm in a Grid," in *GRID*. IEEE, 2008, pp. 340–345.

[22] G. Cordasco, G. Malewicz, and A. L. Rosenberg, "On Scheduling Expansive and Reductive DAGs for Internet-Based Computing," in *ICDCS*. IEEE Computer Society, 2006, p. 29.

[23] ——, "Advances in IC-Scheduling Theory: Scheduling Expansive and Reductive DAGs and Scheduling DAGs via Duality," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 11, pp. 1607–1617, 2007.

[24] ——, "Applying IC-Scheduling Theory to Familiar Classes of Computations," in *IPDPS*. IEEE, 2007, pp. 1–8.

[25] M. Sims, G. Cordasco, and A. L. Rosenberg, "On Clustering Tasks in IC-Optimal DAGs," in *ICPP*. IEEE Computer Society, 2008, pp. 381–388.

[26] G. Cordasco, G. Malewicz, and A. L. Rosenberg, "Extending IC-scheduling via the Sweep Algorithm," *J. Parallel Distrib. Comput.*, vol. 70, no. 3, pp. 201–211, 2010.

[27] G. Cordasco, A. L. Rosenberg, and M. Sims, "On Clustering DAGs for Task-Hungry Computing Platforms," *Central Europ. J. Computer Science*, vol. 1, no. 1, pp. 19–35, 2011.

[28] G. Cordasco and A. L. Rosenberg, "Area-Maximizing Schedules for Series-Parallel DAGs," in *Euro-Par (2)*, ser. Lecture Notes in Computer Science, P. D'Ambra, M. R. Guarracino, and D. Talia, Eds., vol. 6272. Springer, 2010, pp. 380–392.

[29] G. Cordasco, R. D. Chiara, and A. L. Rosenberg, "On Scheduling DAGs for Volatile Computing Platforms: Area-Maximizing Schedules," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1347–1360, 2012.

[30] G. Cordasco and A. L. Rosenberg, "On scheduling series-parallel DAGs to maximize AREA," *Int. J. Found. Comput. Sci.*, vol. 25, no. 5, pp. 597–621, 2014.

[31] S. T. Roche, A. L. Rosenberg, and R. Rajaraman, "On Constructing DAG-Schedules with Large AREAs," in *Euro-Par 2014 Parallel Processing*, F. Silva, I. Dutra, and V. S. Costa, Eds. Secaucus, NJ, USA: Springer International Publishing, 2014, pp. 620–631.