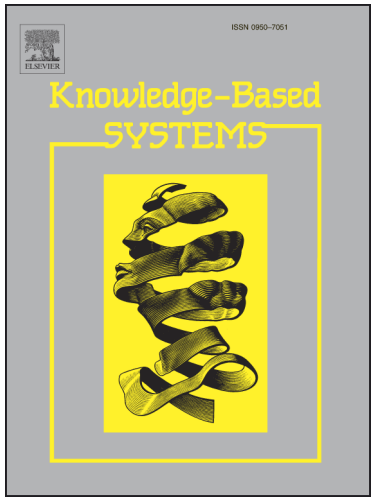
Accepted Manuscript



Resource-Efficient Workflow Scheduling in Clouds

Young Choon Lee, Hyuck Han, Albert Y. Zomaya, Mazin Yousif

PII: S0950-7051(15)00055-6
DOI: http://dx.doi.org/10.1016/j.knosys.2015.02.012
Reference: KNOSYS 3073

To appear in: *Knowledge-Based Systems*

Received Date: 31 October 2014
Revised Date: 11 February 2015
Accepted Date: 11 February 2015

Please cite this article as: Y.C. Lee, H. Han, A.Y. Zomaya, M. Yousif, Resource-Efficient Workflow Scheduling in Clouds, *Knowledge-Based Systems* (2015), doi: http://dx.doi.org/10.1016/j.knosys.2015.02.012

Resource-Efficient Workflow Scheduling in Clouds

Young Choon Lee[a], Hyuck Han[b], Albert Y. Zomaya[a], Mazin Yousif[c]

[a] *The University of Sydney, Australia*
{*young.lee, albert.zomaya*}*@sydney.edu.au*
[b] *Dongduk Women's University, Korea*
hhyuck96@dongduk.ac.kr
[c] *T-Systems International*
myousif100@gmail.com

Abstract

Workflow applications in science and engineering have steadily increased in variety and scale. Coinciding with this increase has been the relentless effort to improve the performance of these applications through exploiting the abundance of resources in hyper-scale clouds and with little attention to resources efficiency. The inefficient use of resources when executing scientific workflows results from both the excessive amount of resources provisioned and the wastage from unused resources among task runs. In this paper, we address the problem of resource-efficient workflow scheduling. To this end, we present the Maximum Effective Reduction (MER) algorithm, a resource efficiency solution that optimizes the resource usage of a workflow schedule generated by any particular scheduling algorithm. MER trades the minimal makespan increase for the maximal resource usage reduction by consolidating tasks with the exploitation of resource inefficiency in the original workflow schedule. The main novelty of MER lies in its identification of "near-optimal" trade-off point between makespan increase and resource usage reduction. Finding such a point is of great practical importance and can lead to: (1) improvements in resource utilization, (2) reductions in resource provisioning, and (3) savings in energy consumption. Another significant contribution of this work is MER's broad applicability. In essence, MER can be applied to any environments that deal with the execution of (scientific) workflows of many precedence-constrained tasks although MER best suits for the IaaS cloud model. Based on results obtained from our extensive simulations using scientific workflow traces, we demonstrate MER is capable of reducing the amount of actual resources used by 54% with an average makespan increase of less than 10%. The efficacy of MER is further verified by results (from a comprehensive set of experiments with varying makespan delay limits) that show the resource usage reduction, makespan increase and the trade-off between them for various workflow applications.

Keywords: Cloud Computing, Scientific Workflows, Resource Efficiency, Resource Management, Workflow Scheduling

1. Introduction

As resouce capacity in clusters and more recently clouds becomes increasingly abundant with the prevalence of large-scale multi-core systems and advances in virtualization technologies, such resource abundance has been "relentlessly" exploited particularly to improve applications performance. Scientific workflows (such as Montage [1], CyberShake [2], LIGO [3], Epigenomics [4] and SIPHT [5]) in particular can take great advantage of abundant resources as they are mostly resource-intensive with a good degree of scalability. However, the exploitation of resource abundance is a double-edged sword in that the performance improvement from such exploitation is often achieved at the expense of resource efficiency.

The inefficient use of resources when executing scientific workflows results from both the excessive amount of resources provisioned and the wastage from unused resources (Figure 1), including idle time[1] among task runs. The optimization of resource efficiency is of great practical importance considering its numerous benefits in the economic and environmental sustainability of large-scale computer systems like corporate data centers and clouds.

While previous work on workflow scheduling has focused on increasing the performance (makespan) with a limited amount of resources (resource scarcity), the advent of multi-core processors and cloud computing (resource abundance) has brought much attention to resource efficiency.[2] Existing workflow scheduling algorithms may be able to adapt to deal with resource abundance by limiting the number of resources to be used (resource limit) at the time of scheduling. However, it is only a partial and ad-hoc solution; for example, the schedule in Figure 1c with a resource limit of 2 shows that makespan increases by 21% compared to the schedule in Figure 1b in return for the use of one less resource. Moreover, the efficacy of such a solution varies for different applications, and even with executions of a particular application with different inputs (e.g., data and/or parameter values). Dynamic resource provisioning with public clouds as in [7, 8] might be an alternative; however, the poor resource utilization—sourced from uneven widths of different levels in a workflow (see Figure 1)—within the hour still remains. Since it is very difficult, if not impossible, to find the optimal resource amount for scheduling a given workflow application, and since current workflow scheduling algorithms perform quite well in terms of makespan, the post-processing of output workflow schedules may be a practical approach to optimizing resource usage.

In this paper, we consider the problem of efficient use of resource abundance for running large-scale scientific workflows. To this end, we develop a new algorithm (*Maximum Effective Reduction* or MER) to effectively trade makespan increase for resource usage reduction. (*Maximum Effective Reduction* or MER), a workflow schedule optimization algorithm that minimizes the resource usage

[1] Non-negligible amount of power is drawn while CPU is idling (e.g., 50% or more of peak power in C1 state)[6].

[2] Resource scarcity and abundance is relative to applications resource requirements.

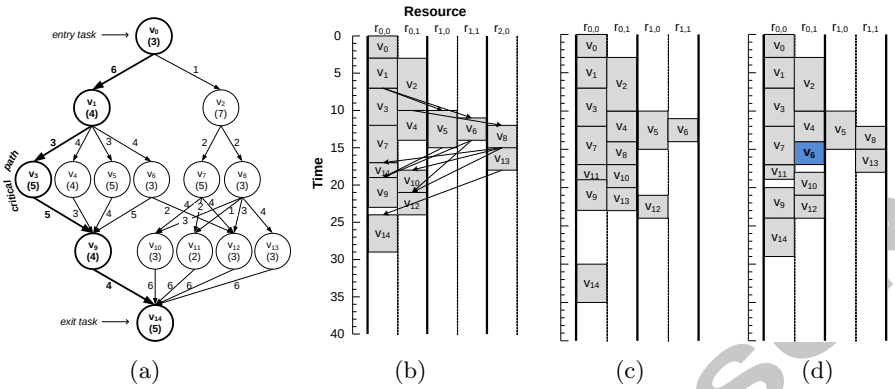


Figure 1: Workflow scheduling example in dual-core compute resources. (a) A sample workflow. (b) The schedule generated using CPF in our previous work, [9]. (c) An example schedule with a pre-determined resource limit of 2. (d) A compacted schedule preserving the makespan of the original schedule, (the schedule in (a) in this example) with the use of the primitive solution in [9]. Computation costs and communication costs are shown in parentheses of vertices and long edges, respectively. Inter-task communications are only present (arrows in Figure 1b) when they are scheduled on different resources as on-chip communication is negligible, i.e., 0.

of a workflow schedule generated by any particular scheduling algorithm. MER is a post-optimization algorithm that takes an existing workflow schedule and consolidates tasks in the schedule into a smaller amount of resources compared to that used for the input schedule. The algorithm aims to find a minimal makespan increase that reduces the resource usage the most; this reduction is defined as effective reduction (ER) in this paper. ER is used to measure resource efficiency based primarily on the difference between the resource usage reduction and makespan increase in a resultant schedule as compared to the input schedule.

In our previous work [9], we have shown the potential that the resource usage of workflow schedule can be reduced by consolidating tasks exploiting idle/inefficiency slots; however, the degree of such reduction is mostly limited due to the preservation of makespan (Figure 1d). MER significantly extends the primitive solution in [9] by allowing makespan increase/delay to maximize resource usage reduction. In particular, MER incorporates a simple, yet effective heuristic to estimate the makespan increase given the tasks being considered for consolidation. The estimate is used for the degree of delay allowed in makespan (makespan delay limit or simply delay limit), and is based on the estimated ER. We have verified the efficacy of this heuristic by a comparison with the best empirical delay limits found in our experiments.

Based on results obtained from our extensive simulations using scientific workflow traces, we demonstrate MER is capable of reducing the amount of actual resources used by 54% with an average makespan increase of less than 10%. The efficacy of MER is further verified by results (from a comprehensive set of experiments with varying makespan delay limits) that show the resource

usage reduction, makespan increase and the trade-off between them for various workflow applications.

The rest of this paper is organized as follows. Section 2 describes our workflow schedule optimization problem. In Section 3, we detail our solution algorithm. Section 4 demonstrates the efficacy of MER with results obtained from our extensive simulations. Section 5 provides a suvery on related work followed by our conclusion in Section 6.

2. The Problem of Workflow Schedule Optimization

In this section, we define the workflow schedule optimization problem describing workflow and system models.

2.1. Scientific Workflows

A scientific workflow consists of a set of precedence-constrained tasks represented by a directed acyclic graph (DAG), $G = (V, E)$ comprising a set V of tasks, $V = \{v_0, v_1, ..., v_n\}$, and a set E of edges, each of which connects two tasks representing their precedence constraint or data dependency (Figure 1a). A task is regarded as ready to run (or simply as a 'ready task'). The readiness of task v_i is determined by its predecessors (parent tasks), more specifically the one that completes the communication at the latest time. More formally, the earliest start and finish times of a task v_i are defined as:

$$EST(v_i) = \begin{cases} 0 & \text{if } v_i = v_{entry} \\ \max\limits_{v_p \in P_i} \{EST(v_p) + w_p + c_{p,i}\} & \text{otherwise,} \end{cases} \quad (1)$$

$$EFT(v_i) = EST(v_i) + w_i \quad (2)$$

where v_{entry} is an entry task without any predecessors, P_i is the set of parent tasks of v_i, w_p and w_i are the computation cost (execution time) of v_p and v_i, respectively, and $c_{p,i}$ is the communication cost from v_p to v_i.

The latest start and finish times of v_i are then defined as:

$$LST(v_i) = LFT(v_i) - w_i, \quad (3)$$

$$LFT(v_i) = \begin{cases} EFT(v_i) & \text{if } v_i = v_{exit} \\ \min\limits_{v_c \in C_i} \{LST(v_c) - c_{i,c}\} & \text{otherwise} \end{cases} \quad (4)$$

where C_i is set of child tasks of v_i and v_{exit} is an exit task.

The actual start and finish times of a task v_i are denoted as $AST(v_i)$ and $AFT(v_i)$, and they can be different from its earliest start and finish times if the actual finish time of another task scheduled on the same resource is later than $EST(v_i)$. $ALST(v_i)$ and $ALFT(v_i)$ are defined in the same fashion. The LFT (or $ALFT$) of v_i is often denoted as the task deadline; that is, a delay

beyond this time increases the makespan. Thus, the maximum delay without a makespan increase (or allowable delay) is defined as the difference between LFT and EFT (or $ALFT$ and AFT) of v_i; this allowable delay is present as a result of synchronization required for child tasks of v_i.

We assume characteristics of a workflow, including the composition, data and computational requirements, are either known or can be obtained using application profiling and performance estimation techniques, e.g., [4, 10].

2.2. Target Systems

The target system in this study consists of a set R of homogeneous compute resources, $R = \{R_0, R_1, ..., R_m\}$. A resource can be a physical compute node or a virtual machine (particularly in public clouds). Each resource in R consists of a set of p processing elements or (virtual) processor cores, i.e., $R_i = \{r_{i,0}, r_{i,1}, ..., r_{i,p}\}$. We assume inter-task communication costs are negligible (i.e., 0) within a single resource. Resources are assumed to be homogeneous in terms of their core count, computing power and cost.

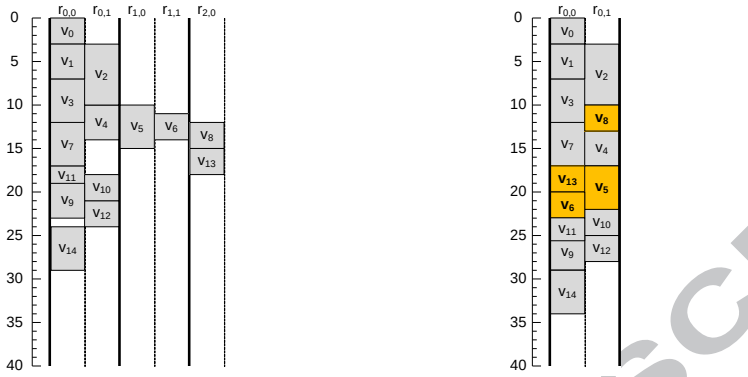
2.3. Problem Formulation

Assume a workflow schedule S^0 is the output of a scheduling algorithm for a given scientific workflow G and a set of resources R. A schedule can be represented as a Gantt chart, e.g., Figure 1b. The output schedule is an execution plan for tasks in G with a subset R^0 of resources used from R, and it contains a set of 3-tuples. Each 3-tuple consists of a task v_i, a resource $r_{j,k}$ and $AST(v_i)$. A set of tasks scheduled on resource $r_{j,k}$ is denoted as $V_{j,k}$. The total resource time used for resource $r_{j,k}$ is defined as the summation of execution times of tasks in $V_{j,k}$, and is denoted by $RT(r_{j,k})$.

The workflow schedule optimization problem addressed in this paper is to find a consolidated schedule S^*, of an original output schedule S^0[3], that maximizes the reduction in resource usage (i.e., the number of resources used or $|R^0|$) with the minimal makespan increase. However, when a delay to makespan is considered, the workflow schedule optimization becomes much more complex. In particular, a delay to the completion time of a task (and possibly the makespan) caused by a consolidation operation may recursively propagate to a number of tasks including not only its successor tasks, but also tasks scheduled after the consolidated task and their successor tasks.This effect is referred to as *delay propagation*.

The goal of workflow schedule optimization is therefore to find the best trade-off between makespan delay and resource usage reduction. To this end, we adopt effective resource usage reduction, or simply *effective reduction (ER)* to determine the actual reduction in resource usage (RUR) considering the

[3] S^0 is the main input and is generated by any given scheduling algorithm.



(a) Original schedule by CPF. (b) Optimized schedule by MER.

Figure 2: Workflow schedule optimization. The original schedule in (a) has a makespan of 29 with the use of three dual-core resources whereas the optimized schedule using MER in (b) has a makespan of 34 (17% increase) with the use of one resource (67% reduction); hence, ER: 0.5 or 50%.

increase in makespan (MI). The effective reduction (ER) of a resultant schedule is defined as

$$ER = \frac{(|R^0| - |R^*|)}{|R^0|} - \frac{(ms^* - ms^0)}{ms^0}, \tag{5}$$

where $|R^*|$ is the amount of resources used in the resultant schedule, ms^0 is the makespan of input schedule and ms^* is the makespan of resultant schedule.

3. Maximum Effective Reduction Algorithm

The Maximum Effective Reduction (MER) algorithm is an workflow schedule optimization technique for workflow scheduling algorithms and it optimizes the trade-off between makespan increase and resource usage reduction. MER consists of the following phases/sub-algorithms: (1) delay limit identification, (2) task consolidation and (3) resource consolidation.

In essence, MER seeks to optimize a workflow schedule by consolidating tasks in two ways: (1) filling idle time slots resulting from data dependencies between tasks and (2) squeezing in between tasks. In general, resources with few tasks are considered for consolidation and tasks are shifted to other resources if this shifting improves resource efficiency. The degree of consolidation can be adjusted by modifying the delay allowed in makespan (delay limit). Specifically, MER enables a task to be consolidated by pushing down one or more tasks within a delay limit to the original makespan (Figure 2b). In the case of public cloud deployment, a delay limit might be set as a multiple of the accountable time unit (e.g., an hour in Amazon EC2) to increase cost efficiency.

Algorithm 1: Delay Limit Identification

```
1  R' ← sorted resources of R^0 in ascending order by RT
2  group R' by RT
3  srcnt ← 0
4  ER^max ← 0
5  for R'_i ∈ (R' − R'_|R'|) do
6      srcnt ← srcnt + |R'_i|
7      ms' ← ms^0 + RT'_i
8      trcnt ← |R^0| − srcnt
9      if srcnt > trcnt then
10         srcnt' ← (⌈srcnt/trcnt⌉ − 1) · trcnt
11         while srcnt' > 0 do
12             ms' ← ms' + srcnt'^th RT in R'
13             srcnt' ← srcnt' − trcnt
14     MI ← (ms' − ms^0)/ms^0
15     RUR ← srcnt/|R^0|
16     ER ← RUR − MI
17     if ER > ER^max then
18         ER^max ← ER
19         d^limit ← MI
```

3.1. Delay Limit Identification

For a given input schedule, MER consolidates it allowing a certain degree of makespan increase; and, thus the identification of the minimum degree of such increase that maximally reduces resource usage is crucial. Our solution to this is the *Delay Limit Identification* algorithm (Algorithm 1). The rationale behind this algorithm is that consolidating resources with a small RT (total resource time used) affects a lesser degree of makespan increase.

Resources used in the input schedule (R^0) are sorted (R') in increasing order and grouped (denoted as R'_i) by total resource time used (RT). The outer `for` loop (line 5) iterates each of those sorted resource groups to identify the maximum effective reduction. The total resource time used by tasks on a particular resource (R'_i in line 7) is a good indicator to dictates the maximum makespan increase if the resource is consolidated. Since we deal with resource groups, the effective reduction for any particular consolidation of resources in a target resource group is maintained considering those in the target resource group and those in previous resource groups (line 6). The numbers of resources to be considered for consolidating their tasks to other resources and for being consolidated are denoted as *srcnt* and *trcnt*, respectively; and thus, the reduction of resource usage (RUR) is determined by *srcnt* (line 15).

In Figure 2a, RTs of three resources are 45, 8 and 6 from R^0_0 to R^0_2, respectively. As stated in Section 2.3, RT for a particular resource is calculated by

Algorithm 2: Task Consolidation

```
1  R* ← R^0 in reverse order
2  S* ← S^0
3  for r*_i ∈ R* do
4      R' ← R* − r*_i
5      for v*_{i,j} ∈ V*_i do
6          for r'_k ∈ R' do
7              AST(v*_{i,j}, r'_k) ← FindMinMISlot(v*_{i,j}, r'_k)
8              if AST(v*_{i,j}, r'_k) ≠ ∞ then
9                  insert v*_{i,j} at AST(v*_{i,j}, r'_k)
10                 update schedule (S*)
11                 d^limit ← d^limit − mi^min
12                 break
13     if r*_i == ∅ then
14         R* ← R* − r*_i
```

summing up execution times of all tasks. For instance, the RT of R_0^0 in Figure 2a is the sum of execution times of all 11 tasks across two cores, i.e., v_0, v_1, $\cdots$, v_{14} in $r_{0,0}$ and v_2, v_4, $\cdots$, v_{12} in $r_{0,1}$. The use of RT is due to the fact that in the worst case scenario all these 11 tasks are consolidated into a single core. In our example in Figure 2, the order of resources by RT is R_2^0, R_1^0 and R_0^0; the actual order after sorting then follows R'_0 and R'_1, except R'_2 (i.e., $R' - R'_{|R'|}$ in line 5). In the very first iteration, values of $srcnt$, $trcnt$, RT'_i and ms' are 1, 2, 6 and 35, respectively. Then, MI, RUR and ER are 0.21 (21%), 0.33 (33%) and 0.12 (12%), respectively.

Makespan increase (MI) may be determined recursively applying RT of source resources ($srcnt$) if it is larger than $trcnt$ (line 9). In our example in Figure 2a, source resources in such a calculation are simply 2nd and 1st resources in R'. Then, the calculation follows $ms' = 29 + 6 + 8$ and $srcnt = 2$, and MI, RUR and ER are 0.48, 0.67 and 0.19, respectively; hence, the schedule in Figure 2b. The ER value of 0.19 is set as the delay limit as evident in the makespan of optimized schedule in Figure 2b, i.e., 34, an increase of 17% in the original makespan of 29.

At the end of each iteration, the maximum effective resource (ER^{max}) is compared with the current ER to identify the ultimate ER^{max} that is the delay limit (d^{limit}) used for the task consolidation algorithm/phase (Algorithm 2).

3.2. Task Consolidation

In essence, a subset of resources for which the makespan increase bounded by the delay limit identified using Algorithm 1 after their consolidation with the rest of resources is considered for task consolidation (Algorithm 2). As shown in line 1, resources used (R^0) for the input schedule (S^0) are first rearranged in

Algorithm 3: Find Minimum Makespan Increase Slot

```
1  function FindMinMISlot(v*_{i,j}, r'_k)
2    MI^min ← ∞
3    for v'_{k,l} ∈ V'_k do
4      calculate AST(v*i, j, r'_k) before v'_{k,l}
5      if AST(v*i, j, r'_k) > LST(v*i, j, r'_k) + d^limit then
6        break
7      d^max ← AST(v*i, j, r'_k) − LST(v*i, j, r'_k)
8      v^cur ← v'_{k,l}
9      v^next ← v^cur + 1
10     while v^next ≠ ∅ do
11       if AFT(v^cur, r'_k) > AST(v^next, r'_k) then
12         d^cur ← AFT(v^cur, r'_k) − LST(v^next, r'_k)
13         if d^cur > d^max then
14           d^max ← d^cur
15         else
16           break
17       v^next ← v^cur + 1
18     MI ← d^max / ms^0
19     if MI < MI^min then
20       MI^min ← MI
21       AST* ← AST(v*i, j, r'_k)
22   if MI^min > d^limit then
23     return ∞
24   else
25     return AST*
```

reverse order as the resource efficiency of a given schedule is diminishing towards the right most resource (Figure 1b). Each ($v^*_{i,j}$) of tasks in a source resource (r^*_i) is then considered for consolidation with one of other resources. The resource on which the consolidation of $v^*_{i,j}$ incurs the minimum makespan increase is found (line 7 using Algorithm 3 and line 8) and $v^*_{i,j}$ is consolidated/inserted to that resource (line 9). Finding the slot for a given task to be inserted with the minimal makespan increase considering the delay limit is rather a complicating task. To this end, we devise `FindMinMISlot` (Algorithm 3). The latest start time (LST, Equation 3) is used at the core of this algorithm as it determines the degree of makespan increase.

After the consolidation of a particular task, the schedule and delay limit are updated (lines 10 and 11). Updating schedule (line 10) may deal with more than several tasks if the consolidated task makes a delay beyond the LST of

Algorithm 4: Resource Consolidation

```
1  R' ← sorted resources of R* in ascending order by #used cores
2  R' ← R'− resources fully used
3  R'' ← R' - resources with #used cores > total#cores / 2
4  for r''_i ∈ R'' do
5      R' ← R' − r''_i
6      for r'_j ∈ R' (from the last res) do
7          if #unused cores of r'_j ≥ #used cores of r''_i then
8              consolidate r''_i into r'_j
9              break
10     if r''_i is not consolidated then
11         break
```

any subsequently scheduled tasks or successor tasks (i.e., delay propagation). The delay beyond LST causes some makespan increase; and this needs to be dealt with and such an increase must be limited by the delay limit (the `while` loop in lines 10-17). Specifically, the process is as follows. Tasks affected (delayed their completion beyond LST) by the consolidation are identified and their schedule data are updated. Such an identification takes place recursively from successor tasks of the consolidated task, subsequently scheduled tasks on the same resource of the consolidated task, successor tasks of these subsequently scheduled tasks, and so on. The delay limit (d^{limit}) is then reduced by the current makespan increase (line 11). At the end of Algorithm 3, if the minimum makespan increase (MI^{min}) is less than the delay limit, the time at which the task of interest (being considered for consolidation) is inserted is returned (line 25); otherwise, the infinity is returned to indicate the failure of finding available slot for insertion.

Once all tasks in a particular resource are finished for consolidation and the resource becomes empty after consolidation of its tasks (line 13 in Algorithm 2), we remove it for any further process.

3.3. Resource Consolidation

As a resource in our study can be seen as a server node (or a resource/virtual instance in cloud computing terminology), there are one or more processing elements (or processor cores). Some of resources used in the input schedule might not be fully used in the sense that one or more cores have not been allocated to any tasks. Algorithm 4 deals with those partly used resources by consolidating these resources in a best-fit fashion; this consolidation helps further improve resource efficiency as the number of resources used in the final output schedule can be reduced. Resources are first sorted in increasing order by the number of used cores (line 1). For the sake of algorithmic efficiency, we consider moving tasks of resources, for which the number of used cores is half or

Table 1: Workflow Traces. Each workflow job consists of 20 variants with different characteristics. Epigenomics workflow traces contain additional 120 random jobs in addition to 320 jobs that can be derived with those details in this table.

application	#workflow jobs	#tasks in a job (workflow size)
CyberShake	220	50 & [100, 1000] with an interval of 100
Epigenomics	440	50, [100, 1000] with an interval of 100 and {2000, 3000, 4000, 5000, 6000}
LIGO	220	50 & [100, 1000] with an interval of 100
Montage	220	50 & [100, 1000] with an interval of 100
SIPHT	320	50, [100, 1000] with an interval of 100 and {2000, 3000, 4000, 5000, 6000}

fewer of the total number of cores (lightly used), to more heavily used resources (line 3). Then, resources in the lightly used resource set (R'') are considered for being consolidated with heavily used resources (R').

Resource consolidation is terminated once a lightly resource (r_i'') cannot be consolidated (line 10) because other lightly used resources that have not been processed in R'' have the same or more used cores as they were sorted (line 1).

4. Evaluation

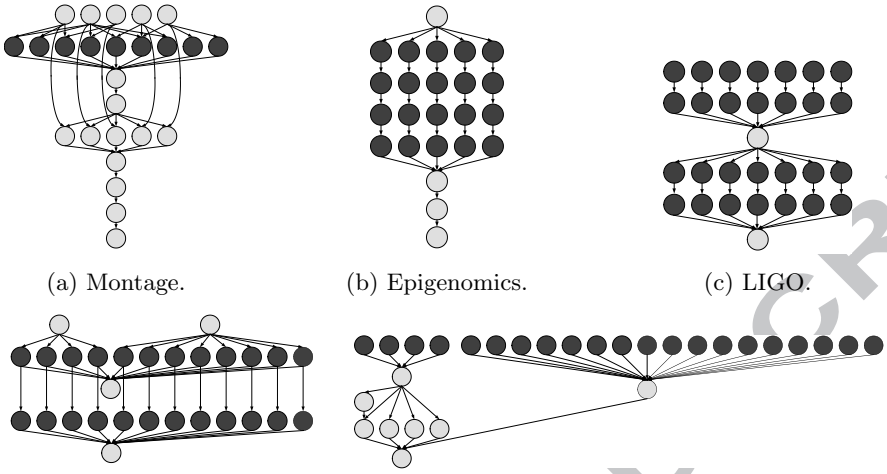
In this section, we evaluate MER with four different scheduling algorithms and under five different workflow applications. The scheduling algorithms we examine are Critical Path First (CPF) [9], Dynamic Critical Path (DCP) [11], Critical-Path-on-a-Processor (CPOP) [12] and a greedy algorithm based on earliest finish time (EFT). We have implemented CPOP and EFT with a slight modification in order to run with an unlimited number of resources. Since the performance of scheduling algorithms is not the focus of this study, these four algorithms were selected arbitrarily. The workflow applications are CyberShake, Epigenomics, LIGO, Montage and SIPHT (Figure 3). Workflow data was obtained from the Pegasus Workflow repository (`https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`) and is summarized in Table 1. The inter-node/resource bandwidth is set to 1 Gbps.

We first show the performance of MER in terms essentially of effective reduction, ER. The efficacy of delay limit identification in MER is then verified by results from an extensive set of experiments conducted with manually-set varying delay limits. Finally, we discuss the optimization overhead of MER.

Although our experiments were carried out with four different resource sizes (#cores: 1, 2, 4 and 8), we only present results obtained with the resource size of 8 due to the similarity of results among resource sizes.

4.1. Performance of MER

Our evaluation results are shown (Table 2 and Figure 4) in terms of makespan increase (MI), resource usage reduction (RUR), delay limit (d^{limit}) and finally



(a) Montage. (b) Epigenomics. (c) LIGO.

(d) CyberShake. (e) SIPHT.

Figure 3: Scientific Workflows.

effective reduction (ER).

MER is proven to perform well with the minimal makespan increase (MI) as shown in Table 2. In particular, the degree of MI on average is 10.5% achieving an average RUR of 54% resulting in an average ER of 43.7%. Results also show MI in the final output schedules is within a fraction of distance from the delay limit (d^{limit}) identified by Algorithm 1.

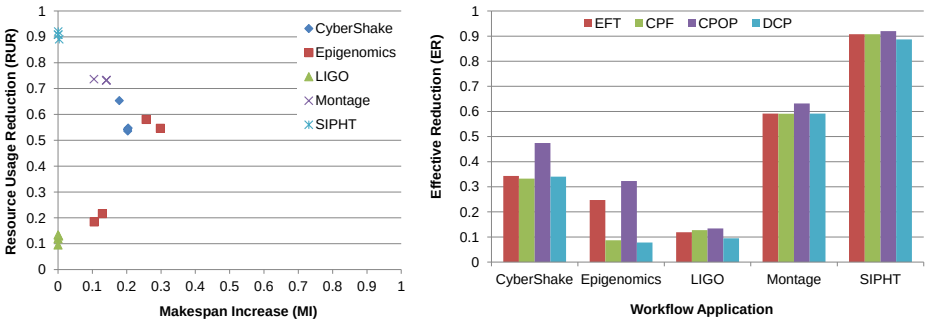
While MER competently demonstrates its efficacy in all cases, it significantly improves resource efficiency for Montage and SIPHT with ER values of at least 59% up to 91%. The main reason for this high degree of resource efficiency improvement lies in the structure of those two workflows, in which a certain level or two have many tasks that can run in parallel leading to excessive use of resources. The efficacy is present with all scheduling algorithms used in our evaluation study.

While Figure 4a shows the relation between MI and RUR, Figure 4b presents the actual ER. Results between different scheduling algorithms are incomparable since ER values are derived from their original output schedules, and the quality of these schedules differs between algorithms. For instance, ER values of CPF and DCP from results for Epigenomics in Figure 4b are lower than those of the other two algorithms; however, this difference is due to the poorer performance of EFT and CPOP in their original schedules (Table 2). While EFT performs comparably to CPF and DCP in terms of makespan, its resource usage ($|R^0|$ of 40.5) is substantially larger than that of CPF and DCP. In the meantime, the higher ER value of CPOP is sourced from both poor makespan and excessive resource usage (ms^0 of 23634 and $|R^0|$ of 40.5 compared with 20875/20878 and 31.2/30.6 of CPF and DCP, respectively). This poor schedule quality implies an inefficient use of resources, leaving many slots idle. A similar pattern of

Table 2: Average of concrete performance results of MER. Results are presented as follows: the makespan (in seconds) of the original schedule (ms^0), the makespan after the consolidation (ms^*) and makespan increase (MI in parentheses) in the first row of each application; the number of resources used in the original schedule ($|R^0|$), the number of resources in the consolidated schedule ($|R^*|$) and resource usage reduction (RUR in parentheses) in the second row; and delay limit (d^{limit}) in the third and last row.

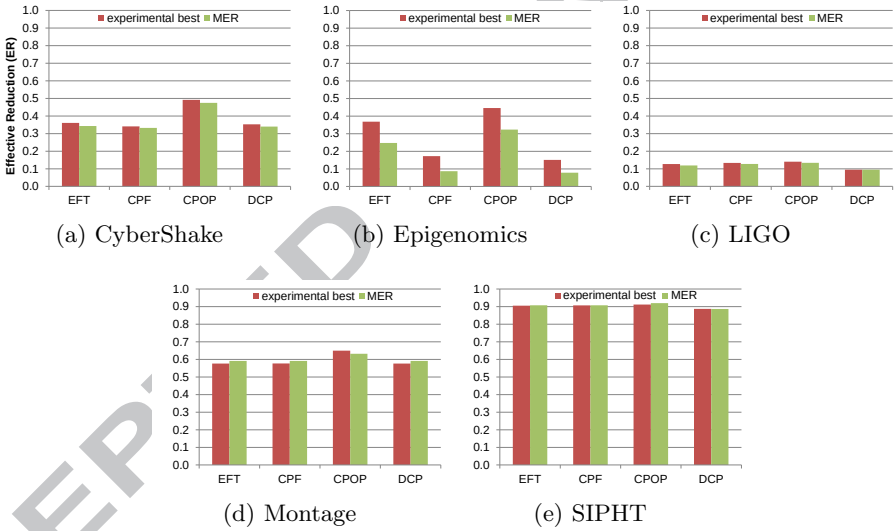
application	EFT	CPF	CPOP	DCP
CyberShake	567, 683 (20%)	566, 681 (20%)	684, 806 (18%)	567, 682 (20%)
	21.0, 9.5 (55%)	20.9, 9.7 (54%)	18.7, 6.5 (65%)	20.8, 9.5 (54%)
	24%	24%	23%	24%
Epigenomics	20878, 27108 (30%)	20876, 23577 (13%)	23634, 29703 (26%)	20878, 23083 (11%)
	40.5, 18.4 (55%)	31.2, 24.4 (22%)	40.5, 17.0 (58%)	30.6, 24.9 (18%)
	39%	18%	36%	14%
LIGO	1398, 1398 (0%)	1398, 1401 (≈0%)	1420, 1420 (0%)	1398, 1398 (0%)
	16.2, 14.2 (12%)	15.4, 13.4 (13%)	16.2, 14.0 (13%)	15.6, 14.2 (9%)
	2%	1%	0%	0%
Montage	212, 241 (14%)	212, 242 (14%)	231, 255 (10%)	212, 241 (14%)
	42.0, 11.3 (73%)	42.0, 11.2 (73%)	41.3, 10.9 (74%)	42.0, 11.3 (73%)
	18%	18%	17%	18%
SIPHT	5169, 5173 (≈0%)	5169, 5170 (≈0%)	7256, 7261 (≈0%)	5169, 5188 (≈0%)
	135.5, 12.4 (91%)	117.2, 10.8 (91%)	135.5, 10.8 (92%)	107.3, 11.8 (89%)
	2%	1%	1%	2%

results can be more clearly observed from Figures 6–10. For example, results of CPOP with "no makespan delay (a delay limit of 0%)" for CyberShake (Figure 6) is more or less equivalent to those of CPF with a delay limit between 20-30% of the makespan. With this delay, however, the resource usage of CPF would be substantially lower (approx. 10) than that of CPOP (12). As MER is a post-processing technique, and is thus independent from scheduling algorithms, results are evaluated and discussed based on the improvement in resource efficiency within each individual scheduling algorithm.



(a) RUR versus MI (b) ER

Figure 4: Performance of MER. Two bottom red squares are results of CPF and DCP, respectively. Their lower RUR compared with that of EFT and CPOP can be justified by a higher resource efficiency in their original schedules as MER exploits inefficiency in the original schedule.



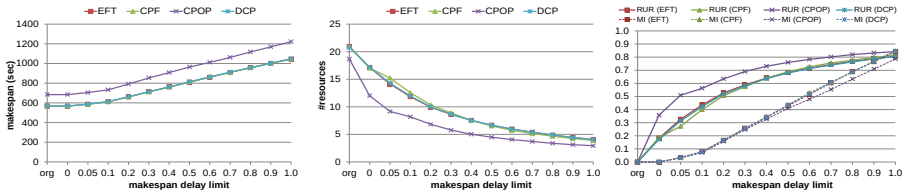
(a) CyberShake (b) Epigenomics (c) LIGO

(d) Montage (e) SIPHT

Figure 5: Performance of MER with respect to experimental best.

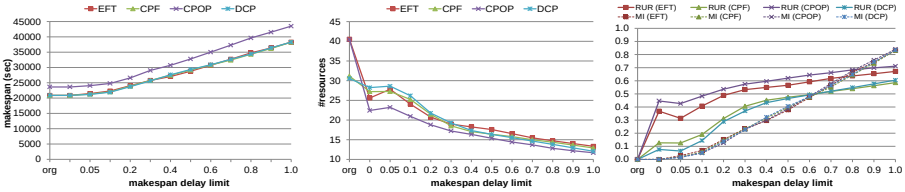
4.2. Experimental Verification

As shown with concrete figures in Table 2, the difference between the actual MI and the delay limit set by MER's delay limit identification algorithm (Algorithm 1) is only a small degree. Although this proves the effectiveness of the algorithm, we further confirm our claim on its effectiveness conducting an extensive set of experiments with twelve varying degrees of delay limits set by hand. In particular, we show that our delay limit identification algorithm is capable of setting the delay limit that is very comparable to experimental best



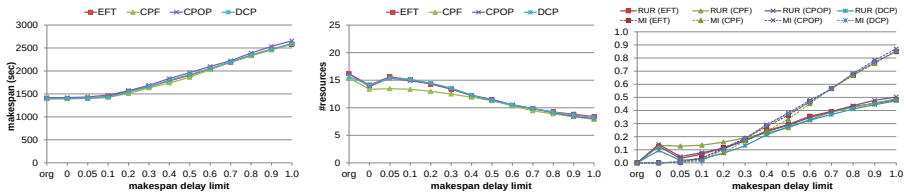
(a) Makespan increase (b) Res. usage reduction (c) RUR versus MI

Figure 6: Performance of MER for CyberShake with respect to manually set delay limits.



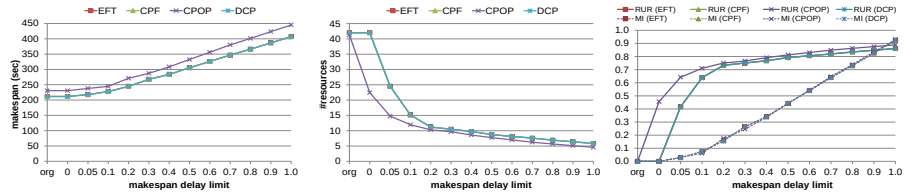
(a) Makespan increase (b) Res. usage reduction (c) RUR versus MI

Figure 7: Performance of MER for Epigenomics with respect to manually set delay limits.



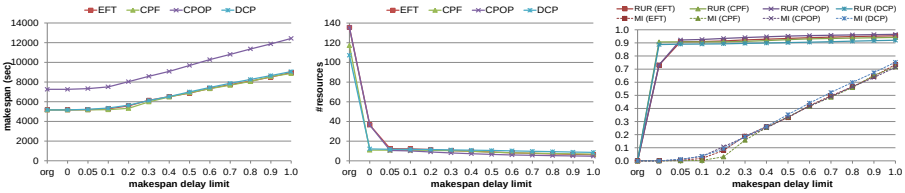
(a) Makespan increase (b) Res. usage reduction (c) RUR versus MI

Figure 8: Performance of MER for LIGO with respect to manually set delay limits.



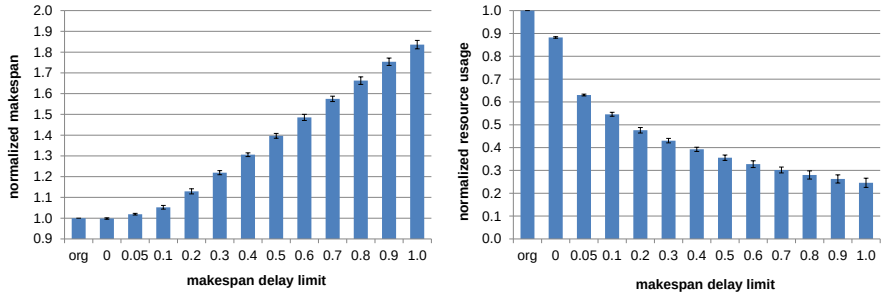
(a) Makespan increase (b) Res. usage reduction (c) RUR versus MI

Figure 9: Performance of MER for Montage with respect to manually set delay limits.



(a) Makespan increase (b) Res. usage reduction (c) RUR versus MI

Figure 10: Performance of MER for SIPHT with respect to manually set delay limits.



(a) Normalized makespan (b) Normalized resource usage

Figure 11: Normalized performance with respect to delay limits. Makespans and resource usage are averaged and normalized based on those of original schedules.

(Figure 5). Detailed experimental results with varying degrees of delay limit are shown in Figures 6–10.

While the maximum ER is mostly achieved with a certain degree of MI, non-negligible amounts of ER are also often observed without MI (Figures 6, 7, 8 and 10); that is, MER achieves a RUR of 12% on average without MI. As can be seen in Figures 6–10, the degree of MI is generally lower than that of RUR; i.e., the slope of RUR, especially with smaller delay limits, is more stiff than that of MI. This fact contributes to the high ER achieved by MER. The overall trends of makespan increases and resource usage reductions are shown in Figures 11a and 11b, respectively.

4.3. Optimization Overhead

In addition to the performance of MER, we have also measured its overhead. On our multi-core machine with 4x10-core Intel Xeon processors, the actual runtimes of MER mostly remain in the millisecond level (between 6 and 58 milliseconds), regardless of workflow type (with the exception of SIPHT, which incurs a runtime of 4.5 seconds). Clearly, overhead is dependent on how intensively consolidation is made. Such a level of overhead is justifiable considering MER's capacity in improving resource efficiency and the execution time of scientific workflows often being a matter of hours [13, 14].

5. Related Work

In this section, we review previous studies on workflow scheduling and discuss resource efficiency.

The execution of scientific workflows is typically planned and coordinated by schedulers/resource managers (e.g., [15, 16]) particularly with distributed resources. At the core of these schedulers are scheduling algorithms.

Traditionally, workflow scheduling focuses on the minimization of makespan (i.e., high performance) within tightly coupled computer systems like compute

clusters with an exception of grids. Various scheduling approaches including list scheduling and clustering are exploited, e.g., [12, 17]. Critical-path base scheduling is one particularly popular approach to makespan minimization [11, 9]. Clustering-based scheduling is another approach getting much attention in the recent past with the emergence of many data-intensive workflows, such as Montage [17].

The partial critical path (PCP) algorithm in [18] tries to minimize the cost meeting the user defined deadline of a given workflow. The cost is positively related to the computing power of service (resource in our definition). For a given workflow, the algorithm first assigns subdeadlines to critical path tasks and to remaining tasks based on deadlines of critical path tasks. Then, it schedules tasks in the cheapest service that can satisfy the subdeadline constraint. The main difference from our work is the fine-grain accounting of service usage without considering the number of services being used.

More recently with the adoption and prevalence of cloud computing, the trade-off between costs and performance has been extensively studied [7, 19, 20]. Most works on workflow scheduling in clouds study the elasticity of cloud resources, i.e., dynamic resource provisioning for cost minimization in particular. However, such dynamic provisioning still remains mostly in the initial workflow deployment; that is, the full elasticity capability of public clouds including that during the execution of workflows is not well explotied with scientific workflows.

The SCPOR workflow scheduling algorithm in [8] is similar to our work in that it attempts to exploit resource abundance, i.e., resource elasticity in public clouds. It is largely based on CPOP [12] with the main difference being the use of elastic (unlimited) resources. It releases an instance when the idle time of the instance reaches to the predefined threshold.

Although there are several studies conducted formulating workflow scheduling as multi-objective optimization problem [21, 22], resource efficiency exploiting inefficient, idle resource time in the presence of resource abundance has not been addressed.

6. Conclusion

The abundance of cloud resources and their elasticity with a pay-as-you-go pricing model provide great opportunities for scientists and engineers in terms particularly of costs and availability. However, the current "public/commodity cloud solutions are neither designed explicitly taking into account scientific computing, nor levelled with the performance of traditional HPC systems. The performance and cost of a cloud cluster deployment are heavily dependent on how effectively the resource abundance of the cloud is exploited, i.e., the determination of "optimal resource usage contributing to resource efficiency. In this paper, we present a workflow schedule optimization algorithm (MER) that can be used with any existing workflow scheduling algorithm as a post-processing technique. It consists of three major phases to first find the trade-off points between the minimum makespan increase and the maximum resource usage reduction, and to consolidate tasks and resources leading to significant improvement in resource

efficiency. Based on results from our extensive experiments with five real-world scientific workflows confirm our claims. Finally, our study has revealed that by allowing a small degree of makespan increase, such exploitation reduces resource usage far greater than any incurred makespan increase. The resulting improvement in resource efficiency has many implications including cost savings and reduction of energy consumption.

We envision the realization of MER's resource efficiency improvement with the support of both application and resource performance profiling and the development/employment of workflow execution system. As part of this plan, we have put significant efforts on the development of our own "cloud-ready" workflow execution system, called DEWE (Distributed, Elastic Workflow Execution, `https://bitbucket.org/lleslie/dwf/wiki/Home`).

Acknowledgement

Dr. Young Choon Lee would like to acknowledge the support of the Australian Research Council Discovery Early Career Researcher Award Grant DE140101628. Prof. Albert Zomaya would like to acknowledge the support of the Australian Research Council Discovery Grant DP130104591.

References

[1] J. C. Jacob, D. S. e. a. Katz, Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking, Int'l J. Computational Science and Engineering 4 (2) (2009) 73–87.

[2] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, K. Vahi, Cybershake: A physics-based seismic hazard model for southern california, Pure and Applied Geophysics 168 (3-4) (2010) 367–381.

[3] A. Abramovici, W. E. Althouse, et. al., Ligo: The laser interferometer gravitational-wave observatory, Science 256 (5055) (1992) 325–333.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Proceedings of Third Workshop on Workflows in Support of Large-Scale Science (WORKS), 2008, pp. 1–10.

[5] J. Livny, H. Teonadi, M. Livny, M. K. Waldor, High-Throughput, Kingdom-Wide Prediction and Annotation of Bacterial Non-Coding RNAs, PLoS ONE 3 (9) (2008) e3197+.

[6] L. A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37.

[7] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011, pp. 49:1–49:12.

[8] C. Lin, S. Lu, Scpor: An elastic workflow scheduling algorithm for services computing, in: Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011, pp. 1–8.

[9] Y. C. Lee, A. Y. Zomaya, Stretch Out and Compact: Workflow Scheduling with Resource Abundance, in: Proceedings of the International Symposium on Cluster Cloud and the Grid (CCGRID), 2013, pp. 219–226.

[10] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, D. V. Wilcox, Pace–a toolset for the performance prediction of parallel and distributed systems, International Journal of High Performance Computing Applications 14 (3) (2000) 228–251.

[11] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, IEEE Transactions on Parallel and Distributed Systems 7 (5) (1996) 506–521.

[12] H. Topcuouglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

[13] D. S. Katz, J. C. Jacob, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, A comparison of two methods for building astronomical image mosaics on a grid, in: Proceedings of the 2005 International Conference on Parallel Processing Workshops, 2005, pp. 85–94.

[14] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, et al., Cybershake: A physics-based seismic hazard model for southern california, Pure and Applied Geophysics 168 (3-4) (2011) 367–381.

[15] M. Litzkow, M. Livny, M. Mutka, Condor - a hunter of idle workstations, in: Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS), 1988, pp. 104–111.

[16] D. Abramson, R. Sosic, J. Giddy, B. Hall, Nimrod: A tool for performing parameterised simulations using distributed workstations, in: Proceedings of the 4th International Symposium on High Performance Distributed Computing (HPDC), 1995, pp. 112–121.

[17] M. Tanaka, O. Tatebe, Workflow scheduling to minimize data movement using multi-constraint graph partitioning, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2012, pp. 65–72.

[18] S. Abrishami, M. Naghibzadeh, D. H. J. Epema, Cost-driven scheduling of grid workflows using partial critical paths, IEEE Transactions on Parallel and Distributed Systems 23 (8) (2012) 1400–1414.

[19] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012, pp. 22:1–22:11.

[20] L. M. Leslie, Y. C. Lee, P. Lu, A. Y. Zomaya, Exploiting performance and cost diversity in the cloud, in: Proceedings of the 6th IEEE International Conference on Cloud Computing (CLOUD), 2013, pp. 107–114.

[21] H. M. Fard, R. Prodan, T. Fahringer, Multi-objective list scheduling of workflow applications in distributed computing infrastructures, Journal of Parallel and Distributed Computing 74 (3) (2014) 2152–2165.

[22] J. J. Durillo, R. Prodan, Multi-objective workflow scheduling in amazon ec2, Cluster Computing 17 (2) (2014) 169–189.