Profiling the scheduling decisions for handling critical paths in deadline-constrained cloud workflows

Taal, A.; Wang, J.; de Laat, C.; Zhao, Z.

DOI
10.1016/j.future.2019.05.002

Publication date
2019

Document Version
Final published version

Published in
Future Generation Computer Systems

Future Generation Computer Systems 100 (2019) 237–249

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Profiling the scheduling decisions for handling critical paths in deadline-constrained cloud workflows

Arie Taal [a,1], Junchao Wang [a,b,1], Cees de Laat [a], Zhiming Zhao [a,*]

[a] Institute for Informatics, University of Amsterdam, The Netherlands
[b] China National Digital Switching System Engineering and Technological Research and Development Center, China

HIGHLIGHTS

- Virtual Infrastructure planning for time critical applications.
- Different heuristics exhibit different performance on different data sets.
- The effect of greediness in VM assignment for tasks in partial critical paths.
- Systematic profiling approach for analyzing greediness in a heuristic solution.
- Recommendation for choosing suitable infrastructure planning strategies.

ARTICLE INFO

Article history:
Received 13 May 2018
Received in revised form 27 April 2019
Accepted 1 May 2019
Available online 11 May 2019

Keywords:
Cloud computing
Workflow planning
Time critical
Partial critical path
Profiling approach

ABSTRACT

In this paper, we study the scheduling decisions for handling deadline-constrained workflows in the context of planning customized virtual infrastructures in the cloud. We specifically focus on the effects of using different types of greediness in selecting cost-effective virtual machines for the tasks in an application's workflow graph. The profiling procedure followed demonstrates that for the widely used approach of the partial critical path algorithm a greedy version is preferred to a more stringent version under different stress conditions, from tight to loose deadlines. Representative topologies of workflow applications are used to generate sets of task graph scheduling problems. Monitoring the performance of the partial critical path algorithm with different types of greediness reveals which of the topologies tested are difficult to solve under various stress conditions. It turns out that an invalid outcome of a greedy version of the partial critical path algorithm is more susceptible to become valid via a final refinement cycle than a less greedy version. The procedure outlined in this paper will allow for a systematic study of a specific heuristic in a workflow scheduling method to increase its success in infrastructure planning under different deadline conditions and is proposed to be part of a general profiling framework.

1. Introduction

In many scientific and industrial applications, e.g., climate modeling [1], disaster early warning [2], or IoT systems [3], workflows composed of many interdependent processing components are present. These workflows are usually complex due to the strict data dependencies among the different processing components, and the required time constraints or deadlines for finishing execution. Such applications often require a well-customized environment for optimizing the workflow performance to meet all the required time and data constraints. Since cloud computing has been evolving to be an obvious choice for more and more companies and research institutes, interest in accommodating more complex workflows on cloud infrastructure emerges [4].

Cloud environments provide virtualized infrastructures (via Infrastructure-as-a-Service) and allow application developers to establish a customized networked virtual environment for specific application requirements. Via the customized virtual environment, application components can then be fast and elastically deployed. Compared to traditional infrastructures, the virtual infrastructures offered allow users not only to pay per use but also to customize the computing and storage capacity (via selected virtual machines(VM)) [5].

Planning a customized virtual infrastructure for a complex workflow in the cloud faces several challenges. Cloud environments offer customizable infrastructure services (e.g., VMs, storage, and network) at different prices. As the execution time of an application's task heavily depends on the selected infrastructure

* Corresponding author.
E-mail addresses: a.taal@uva.nl (A. Taal), j.wang4@uva.nl (J. Wang), delaat@uva.nl (C. de Laat), z.zhao@uva.nl (Z. Zhao).

[1] Both first two authors contributed equally to this paper.

https://doi.org/10.1016/j.future.2019.05.002

service, it is crucial to optimize the selection of infrastructure services for the tasks of the workflow such that the data and time constraints are met for the lowest cost possible.

In case all the characteristics of the different infrastructure services of a cloud provider are known a priori, e.g. all the task execution times on the different VMs, and the task communication times, the planning of the workflow is considered as an off-line workflow scheduling plan. Scheduling approaches which apply the initial a priori state of the cloud are called static scheduling approaches.

In the static scheduling of a workflow, the application is represented by a Directed Acyclic Graph (DAG), where the different vertices of the graph represent the tasks of the application mapped onto the VMs offered by the cloud provider. There exists a rich history in the study of static scheduling problems on different services as multi-processor systems, the Grid, and the cloud. As already stated by Kwok and Ahmad [6] for the multiprocessor environment, and by Jiang et al. [7] for the Grid, it still holds for the cloud that it is difficult to evaluate the performance of the various scheduling approaches quantitatively. Therefore, a framework suitable to profile a (static) scheduling approach is of importance to come with a quantitative appreciating of a scheduling approach. The framework should be equipped with different sets of workflow examples to discover the kind of workflows the approach has difficulty with; Knowledge does not increase from a series of confirmatory observations, disconfirming instances do [8]. These difficult to solve workflow examples may be found by studying the performance under high-stress conditions, e.g. tight deadline conditions. Another important feature of a profiling framework should be a set of graph metrics or topological indices to discriminate between the workflow graphs the scheduling approach under study cannot solve and those it can solve successfully.

Like common practice in machine learning, the different workflow examples should be divided into a so-called training set and test set. The workflow examples in the training set are used to find those workflow examples difficult to solve by the scheduling approach. After finding a metric that can discriminate between the successful and unsuccessful workflows in the training set, the effectiveness of the found metric can be tested on a test set of workflows constructed according to the metric.

To demonstrate the profiling procedure we selected from the many different workflow scheduling approaches [9] the IC-PCP (IaaS Cloud Partial Critical Paths) [10] algorithm, which we slightly modified. The IC-PCP algorithm is a typical scheduling solution for deadline-constrained cloud workflows, which has been a basis for several other extended works. The basic idea of this approach is to map the tasks of the DAG onto the fastest VMs offered by the cloud provider such that the single deadline of the workflow is met and then recursively assign partial critical paths of the DAG to the cheapest VM possible. During the assignment, the communication latencies between tasks are treated as zero if they are on the same VM. This approach provides a straightforward way to assign VMs to an application with a single deadline.

The profiling procedure proposed in this paper will provide a better understanding of selecting algorithms for planning virtual infrastructure under different conditions of application characteristics. The profiling procedure might reveal the limits of a certain kind of heuristics, for instance, a greedy heuristic versus a less greedy, more stringent heuristic. More steps might be added to the profiling procedure presented below, and contribute to the increase of knowledge about different workflow scheduling approaches.

We organized the paper as follows. In Section 2, we review existing works on static scheduling. In Section 3, we propose our profiling approach of the widely used IC-PCP algorithm and present the data sets for the profiling. In Section 4, we summarize and analyze the profiling results. In Section 5, we conclude the paper and plan our future work.

2. Related work

There exists a large variety of workflow scheduling methods for the cloud. Different authors have constructed a taxonomy to capture this large variety of techniques in classification schema [11–13]. From this large diversity of methods, we restrict to those works that can be designated as static workflow scheduling problems for the cloud. In static scheduling all the characteristics of the individual tasks of a workflow, such as a task's processing and communication time and its synchronization requirements, are known before the workflow's execution. A common characteristic of static scheduling is that the set of tasks in the workflow forms a directed acyclic graph (DAG).

Also, the static scheduling approaches exhibit a vast diversity in optimizing the execution cost and execution time under a strict deadline within a given budget. Zheng and Sakellariou [14] used a Budget-Deadline Constrained heuristic called BHEFT, derived from the widely applied HEFT approach, Heterogeneous Earliest-Finish-Time algorithm [15]. Also inspired by the HEFT algorithm, Arabnejad and Barbosa [16] presented the Predict-Earliest-Finish-Time (PEFT) algorithm. Abrishami et al. [10] presented a scheduling algorithm based on the Partial Critical Path method (IC-PCP). Based on the IC-PCP algorithm Shi et al. [17] proposed a data-aware virtual machine type assigning algorithm in their elastic resource provisioning and task scheduling framework.

Methods applying meta-heuristics encompass methods based on Genetic algorithms; Zhu et al. [18] proposed an Evolutionary Multi-objective Optimization method (EMO), Wang et al. [19] presented a genetic algorithm based workflow Planning Algorithm (MEPA), an extended workflow scheduling with advanced resource planning. Other meta-heuristics applied in a global search to find the best solution of a workflow are a Particle Swarm optimization-based heuristic (PSO) applied by Pandey et al. [20], and an Ant Colony optimization-based heuristic applied by Xue et al. [21].

To benchmark these diverse scheduling methods for the cloud, a set of different workflow problems is needed. As all methods try to find the best solution under a strict deadline, it is of importance how this deadline compares to the longest execution path in the workflow graph or the critical path of the workflow. The closer the deadline to the time of the critical path, these scheduling approaches may exhibit difficulties in their search for a solution of the workflow problem. Furthermore, if specific workflow problems turn out to be a tough problem to tackle by some scheduling approaches under tight deadline conditions, it might be appropriate to look for a metric or topological index to characterize those problematic scheduling problems.

The runtime comparison for different Evolutionary Multi -Objective Workflow Scheduling algorithms in the cloud were studied by Zhu et al. [18] using the Real-World Scientific Workflows from the Pegasus project. Workflows from the Pegasus project [22] are the Montage, Cybershake, Epigenomics, LIGO Inspiral, and Sipht workflows, which are an integral part of the profiling procedure presented below. Juve et al. [23] provided a characterization of the workflows from the Pegasus project. Young Choon Lee et al. [24] studied the resource-efficient workflow scheduling in clouds utilizing the Real-World Scientific Workflows. Most recently Ilyushkin and Epema [25] used the Real-World workflows to profile dynamic and plan-based policies during runtime conditions.

However, the effectiveness of scheduling decisions under tight deadline conditions for different workflow topologies is a subject

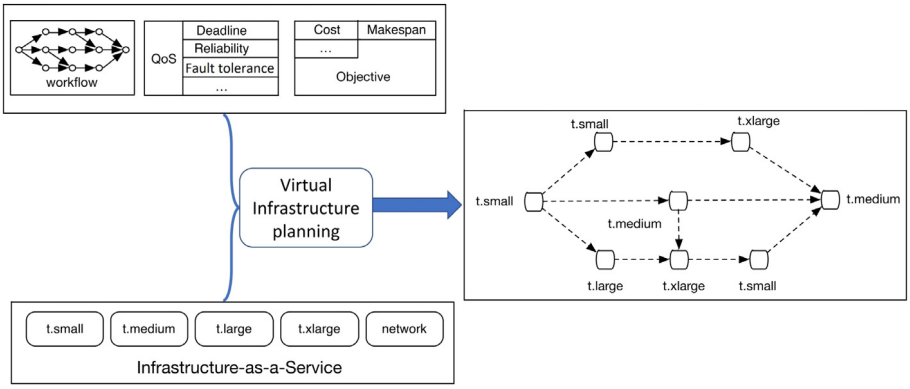


Fig. 1. The infrastructure planning scenario.

not yet addressed in the scientific literature. A systematic profiling approach may reveal whether a chosen heuristic, greedy or less greedy, can be considered as a 'general scheduling solver'.

In the following sections, we present a profiling approach wherein these Real-World workflows are used to pinpoint the effectiveness of different greediness of the IC-PCP scheduling algorithm. This is a new approach to enhance insight into which task graph topologies might be complicated for a scheduling algorithm under tight deadline conditions.

3. Profiling procedure

The basic scenario of virtual infrastructure planning is shown in Fig. 1. It consists of three entities: the cloud provider, the cloud consumer, and the software-defined (virtual) infrastructure. The planning procedure takes the application requirements of a workflow and the available infrastructure resources as input and creates a customized virtual infrastructure for the application. In Fig. 1, the QoS refers to a set of parameters set by the user, reflecting different requirements. The objectives refer to the parameters the user wants to optimize. For instance, some users may want to finish the execution of a workflow as soon as possible. So in such cases, the objective becomes minimizing the workflow's total execution time or the makespan of the workflow. In the scenario of this paper, we try to optimize the overall monetary cost of executing a workflow. The cloud offers different levels of service at different prices which can be defined and provisioned on the user's demand. Such virtual infrastructure services are called "software-defined infrastructures".

To judge how well a scheduling approach performs we define the performance as the total cost of the solution produced by the scheduling approach under the condition that the total execution time of the solution does not exceed the predefined deadline of the workflow. Through our observations, we distinguish the following aspects that may affect the performance of a virtual infrastructure planning approach: the scale of a workflow, the workflow type, deadline, heuristics (greedy or more stringent). The scale of a workflow specifies how many tasks there are inside a workflow, and the workflow type refers to the internal aspects of the DAG, like the degree distribution and dependencies among the different tasks in the DAG. In this paper, we try to characterize the workflow type by a graph metric or topological index.

After all one has to decide which workflow scheduling method(s) one wants to recommend. This choice is not straightforward, as each workflow scheduling approach exhibits weak and strong features. These weak and strong features may become apparent by a systematic profiling procedure that allows studying the effect of the different aspects that influence the virtual infrastructure planning. Fig. 2 depicts a schematic view of a profiling

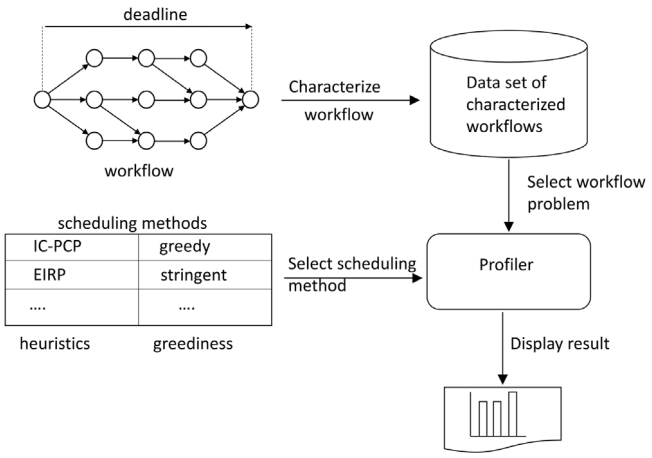


Fig. 2. A schematic view of a profiling procedure.

procedure. By selecting different workflow problems from a data set one can discover which DAGs are difficult to solve by a chosen scheduling method. In case the DAGs of the workflow problems from the data set can be characterized by different metrics or topological indices, one might be able to predict on the basis of these metrics which scheduling approach is the best choice for solving the workflow problem. This is of particular importance in software workbenches that automatically produce a scheduling plan for time-critical applications in the cloud [26,27,4,28,29].

To illustrate the different items of the profiling procedure we select from the diverse set of static scheduling approaches the IC-PCP algorithm [10] by Abrishami et al. to demonstrate the profiling procedure. The IC-PCP algorithm applies a particular kind of heuristics. It tries to assign a path of unassigned tasks in a workflow DAG to a single Virtual Machine (VM). Their approach can be characterized as an adaptation of the Critical Path Method (CPM) for the cloud. The Critical Path Method, developed in the late 1950s by Kelley and Walker ([30,31]), found its way in many different fields of science and technology [32]. In the IC-PCP algorithm, a partial critical path is constructed starting at an unassigned task from which a path of critical parent tasks is constructed. To find the cheapest possible VM for the partial critical path the IC-PCP algorithm applies a greedy assignment criterion. Besides the original greedy assignment criterion, we created a version with a more stringent assignment criterion to study the effect of greediness.

3.1. Problem description

A workflow scheduling problem for an application of tasks $\{t_1, t_2, \ldots, t_n\}$ for a cloud environment is defined by the tuple $WF_C =< S, G, T_{ex}, T_{com} >$, where $S = \{s_1, s_2, \ldots, s_m\}$ is the set of services or virtual machine instances offered by the cloud provider. G is a Directed Acyclic Graph (DAG) representing the task graph of the application associated with the workflow where the set of vertices $V = \{t_1, t_2, \ldots, t_n\}$ represents the set of tasks. The set $E = V \times V$ represents the set of directed edges of the DAG, where $e_{ij} = (t_i, t_j) \in E$ represents the communication between a task t_i and its successor t_j, both in V. The function $T_{ex} : V \times S \rightarrow Z_+$ gives the execution cost $T_{ex}(t_i, s_k)$ of task t_i if executed on service $s_k \in S$, and the function $T_{com} : V \times V \rightarrow Z_+$ gives the communication cost $T_{com}(t_i, t_j)$ between task t_i and its successor t_j.

Furthermore, every DAG has a vertex for an entry task t_0 without any predecessors, and a vertex for an exit task t_{n+1} that does not have any successors. Both tasks have zero execution cost and communication time with their successors and predecessors, respectively.

A valid solution of the workflow scheduling problem is an assignment of services to tasks such that the deadline D of the workflow is not violated. To each task four times are attributed, the earliest start time (EST), the earliest finish time (EFT), the latest start time (LST) and the latest finish time (LFT). These times have to satisfy the following equations for the tasks of the DAG after assigning a service s_k to any task t_i:

$$\begin{aligned} &EST(t_0) = 0,\ EST(t_i) = \max_{t_p \in t_i's\ parents} \{EST(t_p) \\ &+ T_{ex}(t_p, s_k) + T_{com}(t_p, t_i)\}\ i > 0 \\ &EFT(t_i) = EST(t_i) + T_{ex}(t_i, s_k) \\ &LFT(t_{n+1}) = D,\ LFT(t_i) = \min_{t_c \in t_i's\ children} \{LFT(t_c) \\ &- T_{ex}(t_c, s_k) - T_{com}(t_i, t_c)\} \\ &LST(t_i) = LFT(t_i) - T_{ex}(t_i, s_k) \end{aligned} \tag{1}$$

By definition the communication time $T(t_i, t_j)$ is taken zero if both consecutive tasks t_i and t_j are assigned to the same service s_k or VM. The times defined in Eq. (1) express the requirement that a task t_i may not start earlier than any of its parent tasks or predecessor tasks has finished execution and communicating data with the service of t_i.

3.2. Greediness of the IC-PCP algorithm

Applying a critical path-based algorithm is a widely used approach for tackling the multi-processor scheduling problem [33]. These algorithms are designed to solve the task scheduling in a fixed infrastructure (e.g. a fixed number of homogeneous processors). However, the cloud provides infrastructure level programmability, allowing the user to define and provision the resources needed. To such ends, the Partial Critical path approach (IC-PCP) of Abrishami et al. [10] is proposed. This IC-PCP approach resembles according to the classification of greedy algorithms [34] a Best-Local approach. The algorithm starts by assigning the best service, i.e. the fastest most expensive service, to each individual task, the start configuration, with a deadline large enough allowing the start configuration to be valid or implementable. According to the Best-Local approach the IC-PCP algorithm successively assigns tasks of a partial critical path to the cheapest service possible by applying a greedy step with respect to a local optimality criterion, such that each intermediate configuration of services remains the best so far after the greedy step. Besides a local optimality criterion, the algorithm must obey to a global optimality criterion, namely to keep the cost of the final configuration as low as possible, and to realize a latest finish time for the exit task not exceeding the deadline of the workflow. In Algorithms 1, 2 the pseudo code of the IC-PCP algorithm is presented, the variables used in the pseudo code are summarized in Table 1.

Algorithm 1 Parents Assigning Algorithm

```
1:  procedure AssignParents(t)
2:      while t has an unassigned parent do
3:          PCP ← null, t_i ← t
4:          while there exists an unassigned parent of t_i do
5:              add CriticalParent(t_i) to the beginning of PCP
6:              t_i ← CriticalParent(t_i)
7:          call AssignPath(PCP)
8:          for all t_i ∈ PCP do
9:              update EST and EFT for all successors of t_i
10:             update LST and LFT for all predecessors t_i
11:             call AssignParents(t_i)
```

Algorithm 2 Path Assigning Algorithm

```
1: procedure AssignPath(PCP)
2:     s_{i,j} ← the cheapest applicable existing instance for PCP
3:     if s_{i,j} is null then
4:         launch a new instance s_{i,j} of the cheapest service s_i
       which can finish each task of PCP before its LFT
5:     schedule PCP on s_{i,j} and set SS(t_i)
6:     set all tasks of PCP as assigned
```

Before we discuss the performance of the IC-PCP algorithm on different task graph topologies, some clarification is needed about the implementation of the pseudo code. In the Path Assigning Algorithm to find the cheapest possible service (VM) to be assigned to a partial critical path PCP is phrased as "launch a new instance of the cheapest service which can finish each task in a partial critical path PCP before its LFT" [10]. To discover the exact meaning of this phrase it will help to look at the necessary requirement for any possible service $s_i \in S$ for the partial critical path PCP. Suppose $PCP = \{t_m, t_{m+i}, \ldots, t_n\}$, where according to the *AssignParents* procedure in Algorithm 1 critical parents are inserted at the beginning of PCP, so t_{m+i-1} is the critical parent of t_{m+i}. Then an interpretation of the above phrase is that for the cheapest possible service $s_k \in S$ to be assigned to each of the tasks $t_{m+i} \in PCP, i = 0, \ldots, n-m$, the following requirement must hold:

$$\begin{aligned} &EFT(t_{m+i}) = EST(t_m) + \sum_{j=0}^{i} T_{ex}(t_{m+j}, s_k) < LFT(t_{m+i}) \\ &i = 0, \ldots, n-m \end{aligned} \tag{2}$$

It follows that an assignment of the cheapest possible service s_k according to this requirement does not allow for idle time on a VM assigned to a partial critical path, there is no communication time involved between any two consecutive tasks in an assignment. For a workflow with a task graph of n tasks the time complexity of the IC-PCP algorithm is $O(n^2)$ [10].

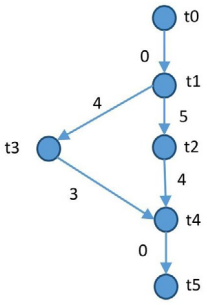
Consider the example task graph shown in Table 2 . In this example t_0 and t_5 are the entry and exit task, respectively, both tasks having zero execution time and communication time.

The start configuration with all unassigned tasks corresponding to VMs of service s_1 results in $EST(t_4) = 23$. After assigning the partial critical path $PCP = \{t_1, t_2, t_4\}$ to a single VM of service s_2, $EST(t_4)$ decreases to 21, as the increase in execution time of t_1 and t_2 for service s_2, equals 7, is overcompensated by zeroing

Table 1
Summary of the variables used in the pseudo code of Algorithms 1, 2.

EST(t_i)	Earliest Start Time, the earliest time task t_i can begin
LST(t_i)	Latest Start Time, the latest time task t_i can begin
s_i	service or virtual machine, one of the services offered by the cloud provider
PCP	Partial critical path, a subgraph of sequential tasks to be scheduled on the same service or virtual machine
$s_{i,j}$	the jth instance of service s_i to be assigned to the tasks in PCP
SS(t_i)	Selected Service, is the service selected for processing task t_i
EFT(t_i)	Earliest Finish Time, the earliest time task t_i may finish, depends on EST(t_i) and the selected service for t_i
LFT(t_i)	Latest Finish Time, the latest time task t_i may finish, depends on LST(t_i) and the selected service for t_i

Table 2
An example task graph to be scheduled on a cloud offering three different services. In the start configuration each task t_i is hosted on a VM of service s_1, the fastest service offered.



Services	t_1	t_2	t_3	t_4
s_1	8	6	7	8
s_2	12	9	10	13
s_3	16	12	13	17

Deadline = 35

Start configuration

	t_0	t_1	t_2	t_3	t_4	t_5
Service	1	1	1	1	1	1
Perf	0	8	6	4	8	0
EST	0	0	13	12	23	31
EFT	0	8	19	19	31	31
LST	4	4	17	17	27	35
LFT	4	12	23	24	35	35

the communication time in the critical path PCP, equals 9. The decreased value $EST(t_4)$ turns out to have severe repercussions in case only Requirement (2) is applied, because this requirement is not violated for this assignment, and consequently, the algorithm stops with an invalid assignment of VMs. According to the *Parents Assigning* Algorithm the unassigned successor t_3 of t_1 gets updated after the assignment of PCP, resulting in $EST(t_3) = 16$ and $EFT(t_3) = 23$, and $LST(t_3)$ and $LFT(t_3)$ are not updated as the algorithm proceeds with *AssignParents*(t_1) because there are no predecessors of t_1 (except entry node t_0). As there are no unassigned parents of t_1, and also no unassigned parents of t_2, the next step in the for-loop of the *Parents Assigning* Algorithm addresses task t_4. Its single unassigned predecessor t_3 gets updated, $LST(t_3) = 12$, and $LFT(t_3) = 19$. Next the algorithm proceeds with *AssignParents*(t_4) not noticing that $LFT(t_3) = 19 < EFT(t_3) = 23$, and fails to assign a VM to t_3 leaving the configuration in a non-implementable state.

One adjustment to make IC-PCP with only Requirement (2) successful for this example is to implement an additional requirement for assigning a service to the tasks of a partial critical path. Observe that t_3 is not a dominant or critical parent of t_4, but should be taken into consideration for a successful ending of the algorithm.

For every member t_{m+i} of *PCP* we determine its dominant parent outside *PCP* designated as $t_{\hat{p};m+i}$, i.e. that parent t_p of all the parents outside *PCP* having the largest sum $EFT(t_p)+T_{com}(t_p, t_{m+i})$, where $T_{com}(t_p, t_{m+i})$ is the communication time between t_p and t_{m+i}. As an additional requirement for assigning any service $s_k \in S$ to the member tasks of *PCP* is:

$$EFT(t_{\hat{p};m+i}) + T_{com}(t_{\hat{p};m+i}, t_{m+i}) \leq EST(t_{m+i}),\ \ i = 0, \ldots, n-m \tag{3}$$

Likewise, for each task t_{m+i} in *PCP* a dominant child $t_{\hat{c};m+i}$ outside *PCP* is determined, which gives the additional requirement:

$$LFT(t_{m+i}) \leq LST(t_{\hat{c};m+i}) - T_{com}(t_{\hat{c};m+i}, t_{m+i}),\ \ i = 0, \ldots, n-m \tag{4}$$

Implementing the IC-PCP algorithm according to Requirements (2)–(4) also produces all the steps of the example workflow presented in [10], but confronted with the example graph in Table 2 the algorithm detects that no assignment for $PCP = \{t_1, t_2, t_4\}$ is possible. Consequently, the final state of the algorithm equals the start configuration.

To find out whether the greedy version with Requirement (2) or the more stringent version with Requirements (2)–(4) is to be preferred, tests on series of task graphs are performed.

The implementation of the IC-PCP algorithm used in section "Profiling results" below differs slightly from the original version in Abrishami et al. [10] (Algorithms 1, 2). We replaced the pseudo code of Algorithm 2 by the pseudo code of Algorithm 3. After assigning a Partial Critical Path *PCP* to a service (procedure *AssignPath(PCP)*), all tasks are updated (whether assigned or unassigned). After that, a recursive call is made for each task in *PCP*.

Algorithm 3 A different implementation of the IC-PCP algorithm according to Abrishami et al. [10] as listed in Algorithm 2.

```
1: procedure AssignPath(PCP)
2:     while there exists an unassigned parent of t_i do
3:         ...
4:     call AssignPath(PCP)
5:     update EST, EFT for all t_k ∈ G
6:     update LST, LFT for all t_k ∈ G
7:     for all t_i ∈ PCP do
8:         call AssignParents(t_i)
```

It turned out that this approach produced better results than an implementation of the original pseudocode from our side. The time complexity has not changed as this version also exhibits the same time complexity as the original IC-PCP algorithm, namely $O(n^2)$. As the focus of this paper is not to benchmark the original IC-PCP algorithm, but to study the success rate of an algorithmic approach under different tight deadline conditions, an implementation with a higher performance is justified. A side effect of this

adjusted IC-PCP algorithm is that in some cases idle time might be introduced in assigned instances of tasks, as not only unassigned but also already assigned task are adjusted. When the example from Table 2 is solved with a larger deadline value equals 39, this adjusted version of the IC-PCP algorithm (applying Requirement (2) only) adds idle time to the instance of service s_2 with tasks t_2, t_4, with the result $EFT(t_2) = 21$ and $EST(t_4) = 26$, resulting in a valid solution. We will address this issue further in section "The greedy version of IC-PCP revisited".

3.3. Data sets

Different topology sets are created to test both versions of the IC-PCP algorithm. From the Pegasus project [22] we took the topologies of the following Real-world Scientific workflow examples. From the Montage examples, part of the NASA/IPAC Infrared Science Archive [35], the examples of Montage 25 and Montage 100 were taken, having 25 and 100 individual vertices, respectively. From the CyberShake workflow examples, used by the Southern California Earthquake Center (SCEC) [36], we choose the Cybershake workflows with 30 and 100 vertices. Epigenomic workflows of 24 vertices and 100 vertices were taken from the Epigenomic workflow examples, originating from the USC Epigenome Center [37]. Inspiral workflows with 30 vertices and 100 vertices were chosen from the LIGO Inspiral Analysis workflow examples, used by the Laser Interferometer Gravitational-Wave Observatory (LIGO) [38]. Lastly, from the Sipht workflow examples, bioinformatics project at Harvard University, Sipht workflows of 30 and 100 vertices were taken.

From each fixed topology example a set of 20 task graphs was generated by assigning execution times to the vertices and communication times to the edges. Execution times are chosen from the integer range $(5, \ldots, 20)$ and represent the time a task will need running on the fastest service s_1. Execution times for the slower services s_2 and s_3 are respectively 2 and 3 times the execution time on service s_1. This is reasonable if compared with numbers from benchmarking VMs [39]. The communication time between any task and a successor is chosen from the integer range (5, 20) under the condition that it may not exceed the execution time of the task on the fastest service. For each task graph, a large enough deadline is generated.

To test a graph metric that discriminates between easy and difficult to solve Real-world Scientific workflow example topologies (see section "Characterization by Graph metrics or Topological indices") with respect to the performance of both versions of the IC-PCP algorithm, a second set of task graphs is generated by means of the open source random graph generator GGen [40]. This tool allows for generating task graphs by controlling properties like in-degree/out-degree of the vertices. Two sets are generated for task graphs having a size of 32 and 64 vertices, where the in-degree parameter has values between 2 and 5 and the out-degree parameter values between 1 and 4. This choice prevents the GGen algorithm from generating simple task graphs, for instance, task graphs with a single path. For each size of a task graph, the graph generated by the GGen algorithm was checked for the metric under study. For a value of the metric indicating the task graph as easy to solve, the graph was added to a set called GG_32_0/GG_64_0. Otherwise, it was added to a set called GG_32_1/GG_64_1. Once each set contains 20 examples, for each generated example computation times and execution times were generated for the different VMs following the same procedure as explained above for the Real-world Scientific workflow examples.

4. Profiling results

In this section we evaluate two implementations of the IC-PCP algorithm for the data sets described above. A greedy version is compared with a less greedy, more stringent version of the IC-PCP algorithm. Experimental results show that under different deadline settings, different greediness can lead to different results.

4.1. Success rate of a greedy versus a more stringent version of the IC-PCP algorithm

The version of the IC-PCP algorithm applied in this section is the adjusted one as explained above, Algorithm 3. Two implementations with respect to assigning a service to a partial critical path are compared with each other, a greedy one and a less greedy one. The greedy one applies the necessary Requirement 2 only in the *Path Assigning* Algorithm and this implementation is designated by PCP_v0. The more stringent implementation applies the additional Requirements (3) and (4), i.e. the Requirements (2)–(4) in the *Path Assigning* Algorithm, and is designated by PCP_v1.

In Fig. 3 the success rate is displayed of the greedy implementation PCP_v0 versus the more stringent implementation PCP_v1 for different topology sets generated from the Real-world Scientific example workflows. All 20 task graphs from a topology set are presented to PCP_v0 and PCP_v1 under different stress conditions, from tight to loose deadlines. The success rate for the 20 task graphs is recorded for each new deadline. A new deadline is related to the time of the critical path of the start configuration, with each task assigned to the fastest service s_1. The new deadline is determined by the stress parameter p and is defined as follows:

$$p = \frac{critical\ path\ time}{deadline} \times 100 \tag{5}$$

In case $p = 100$ the deadline equals the time of the critical path (including communication times), which is the most stressful condition for the algorithm to start with. In case $p = 50$ the deadline is 2 times the time of the critical path. The success rate is recorded for $p \in [100, \ldots, 10]$. In Fig. 3 the success rate is displayed for decreasing values of $p \in \{100, 90, \ldots, 10\}$, i.e. in steps of 10, from tight to loose deadline conditions.

From the success rate for different values of the stress parameter p we can see that there is a difference between the greedy version PCP_v0 and the more stringent version PCP_v1. For almost all topologies, except Montage 25 and Montage 100, the greedy version has higher success rates under tight deadline conditions, $70 < p < 100$, than the more stringent version. As the partial critical path algorithm starts to find the cheapest assignment for tasks of the critical path before it deals with other tasks in the task graph, it is interesting to investigate for which values of the stress parameter p both algorithms are successful in finding the cheapest assignment for the tasks of the critical path.

Fig. 4 displays the success ratio in assigning the critical path for the different topology sets. For the workflow problems in all topology sets it holds that the greedy version PCP_v0 is always able to assign the critical path to a virtual machine in agreement to Requirement (2), but for the Montage topologies it fails to finish successfully under tight deadline conditions, $70 < p < 100$, see Fig. 3. The more stringent version PCP_v1 is able to find an assignment for the critical path in agreement with the additional Requirement (3) and (4) for values $p < 80$. Under tight deadline conditions, $p > 80$, the more stringent version PCP_v1 stops in an early stage where the final solution equals the start configuration with each task assigned to the fastest and most expensive service.

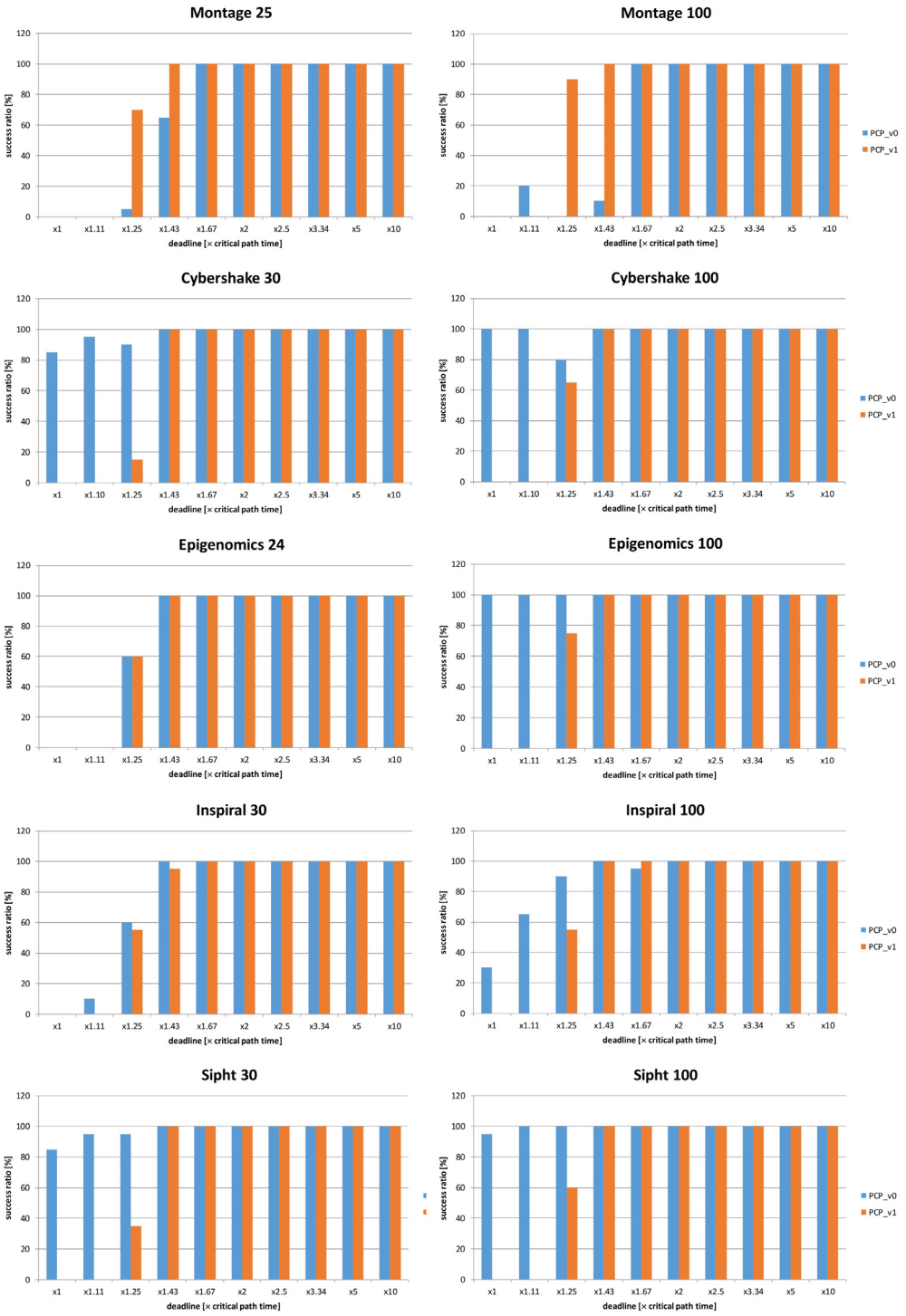


Fig. 3. The success ratio of the greedy algorithm PCP_v0 and the more stringent algorithm PCP_v1 for different topology sets, each set having 20 DAGs. The values $\{\times 1, \times 1.11, \ldots, \times 10\}$ for the deadline correspond with the values $\{100, 90, \ldots, 10\}$ for the stress parameter p.

If we have to choose between the greedy version and the more stringent version according to the success rate, the choice would be the greedy version PCP_v0, which only applies Requirement (2), as only for the Montage topology set there is a preference for PCP_v1 with the extra Requirements (3) and (4).

There is another reason to prefer PCP_v0 above PCP_v1, which will be the subject of the next section "The greedy version of IC-PCP revisited". In that section, we present an extension of the PCP_v0 algorithmic approach that improves the success rate under tight deadline conditions, $70 < p < 100$.

4.2. The greedy version of IC-PCP revisited

The greedy implementation PCP_v0 has an advantage over its more stringent counterpart PCP_v1. In case it fails, i.e. it ends in a final configuration that is not valid because some tasks start too early with respect to one of their parents, the greedy version can benefit from a repair cycle at the end. With a repair cycle we mean a sweep over the DAG after the last assignment taken by the algorithm, in order to resolve a disagreement between the earliest finish time (EFT) of a task and the earliest start time

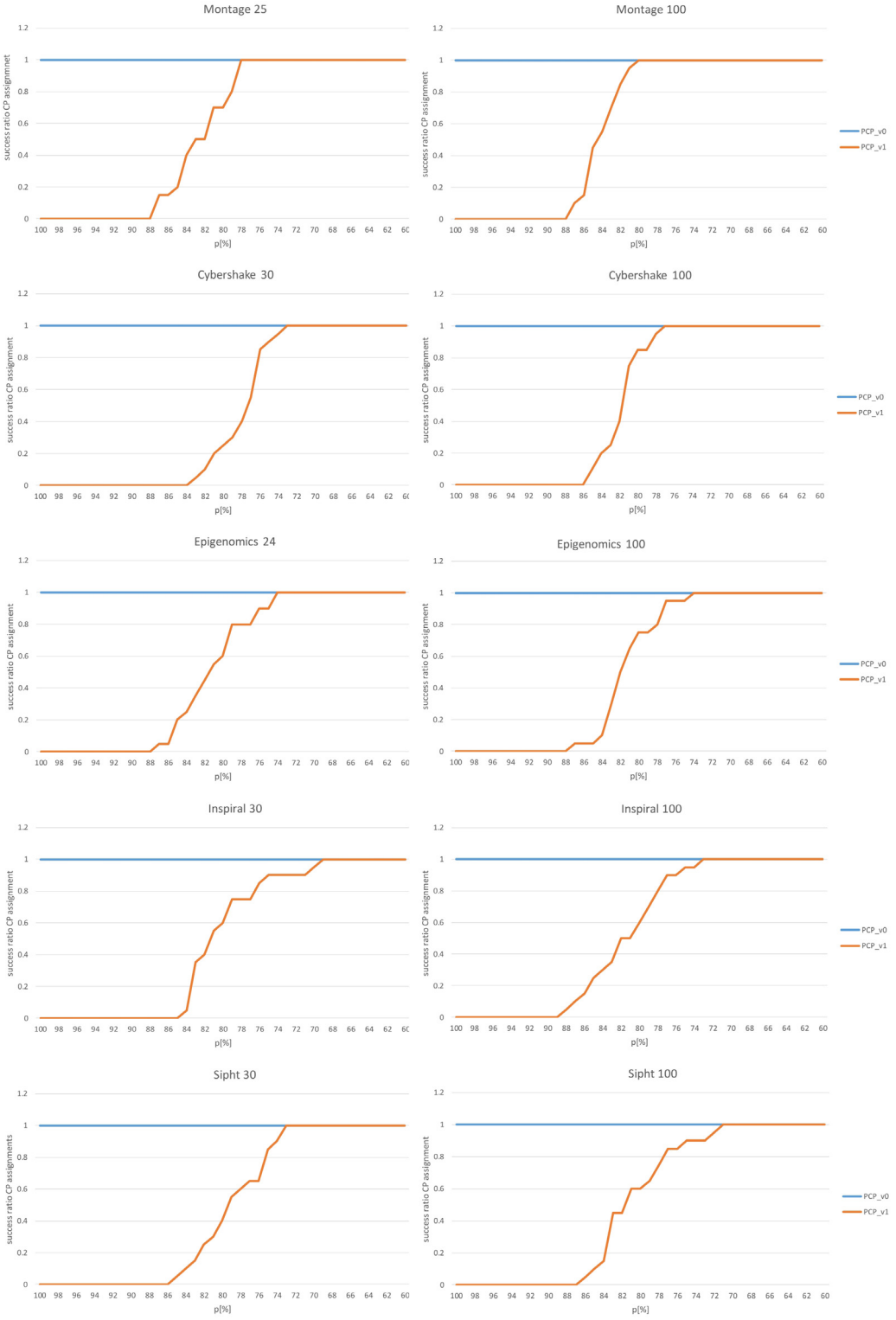


Fig. 4. The success ratio for assigning the Critical Path of the greedy algorithm PCP_v0 and the more stringent algorithm PCP_v1 for the different topology sets.

(EST) of one of its children, likewise for a disagreement between the latest finish time (LFT) of a parent task and the latest start time (LST) of one of its children. If the final configuration is valid, nothing will change. But an invalid configuration might become valid by inserting idle time, i.e. enlarging the EST of a task that starts too early. As explained above, Algorithm 3 may also add idle time to an instance by the updating EST, and EFT of all tasks, after an assignment of a partial critical path. So a valid solution produced by PCP_v0 may also have instances containing idle time. Additionally, to the repair cycle, it is checked whether an instance with idle time can be split into separate instances when the idle time inserted is larger than the original communication time between the adjusted task and its parent task in the instance.

The enhanced implementation of PCP_v0 by adding a repair cycle at the end and a splitting procedure for instances with idle

time is designated as PCP_v01. In Fig. 5 the performance improvement of this enhanced implementation PCP_v01 is compared with the implementation PCP_v0, as applied above in Fig. 3.

As each service or virtual machine has a cost we also looked at the total execution cost of a solution of the scheduling problem. In the same way as in [10] three different services s_1, s_2 and s_3 are distinguished having an arbitrary cost value of 5, 2 and 1, respectively, where we define the cost as the price per unit time value. The higher the cost of a service the faster the execution time for a task. It turns out that a valid solution produced by PCP_v0 also benefits from this splitting procedure, resulting in a lower total execution cost. Another feature of the enhanced implementation PCP_v01 is its better mean total execution cost under tight deadline conditions, $70 < p < 100$, due to the fact that its success rate has been improved. Like an enhanced implementation of the greedy version PCP_v0 we also constructed an enhanced implementation of the more stringent version PCP_v1, designated as PCP_v11 (i.e. PCP_v1 enhanced with a final repair cycle and splitting procedure). In Fig. 6 the mean total execution cost of PCP_v01 is compared with the mean total execution cost of PCP_v11.

The conclusion is that the greedy version with a finishing repair cycle performs well for all topologies except for the Montage topology. As all DAGs for a topology set is generated with the same algorithm, applying the same range of execution times and communication times, the question arises how far certain characteristics of a topology might play a role in predicting the performance under tight deadline conditions. What makes the Montage workflow topologies a difficult topology for the partial critical path algorithm under tight deadline conditions? If one compares the Montage topology with the other topologies [22], one would notice that the Montage topology contains fewer parallel paths. In the next section, we will search for a metric based on the number of parallel paths that allows to discriminate between the Montage topology and the other topologies.

4.3. Characterization by graph metrics or topological indices

When we look at the performance of the greedy version with a finishing repair cycle, PCP_v01, as displayed in Fig. 5, the question arises what the difference is between the Montage topology and the other topologies. As the DAGs for each topology set are generated with the same algorithm it is tempting to look at a pure topological difference, a metric or a topological index, depending on the vertex and edge set only. Here we need a graph metric or a topological index to discriminate between the Montage topologies and the other topologies. The number of metrics and topological indices is huge. From a theoretical perspective, the computational complexity of the metric or topological index is not the most important feature. First, we have to find one that discriminates between the Montage 25/100 topologies and the other topologies in Fig. 5. The problem of an easy to compute metric like the compactness expressed by $E/|V|$ or $E/|V|^2$ is that it does not discriminate well, as two graphs with the same quotient may have a totally different structure. Among the more complex metrics, the name of the metric is not necessarily a good indication of its applicability in case of a task graph. For instance, the hyperbolicity of a graph, much used in the field of network theory, is a measure for the 'Treeness' of a graph, and as such might be useful to characterize a task graph. The more a task graph resembles a tree structure, with a combining entry node, the higher the expected success rate running IC-PCP. However, the hyperbolicity of a graph is not defined for directed acyclic graphs.

As already noticed above the Montage topologies show less parallel paths, which might be an inkling for the lower success rate for these topologies, as displayed in Fig. 5. In Table 3 the values of some easy to calculate metrics are listed for the different topologies applied.

Table 3
The value of different metrics applied to the different workflow topologies.

Metric	$E/\|V\|$	$E/\|V\|^2$	$< outdegree >$	$\#paths$	$\#paths/\|V\|$
Montage 25	1.85	0.07	1.89	90	3.6
Montage 100	2.44	0.02	2.45	1920	19.2
Cybershake 30	1.7	0.05	1.73	26	0.86
Cybershake 100	1.9	0.02	1.91	96	0.96
Epigenomics 24	1.04	0.04	1.12	5	0.20
Epigenomics 100	1.2	0.01	1.22	24	0.24
Inspiral 30	1.31	0.04	1.34	49	1.63
Inspiral 100	1.41	0.01	1.42	218	2.18
Sipht 30	1.74	0.06	1.77	41	1.36
Sipht 100	1.85	0.02	1.86	133	1.33

From the values in Table 3 the number of paths per task in the DAG, $\#paths/|V|$, discriminates between the Montage topologies and the other topologies. As a first choice, we take a value of 2.5 for this metric, below 3.6 (Montage 25) but above 2.18 (Inspiral 100). A first conjecture might be that if a workflow topology has a value below 2.5 we might expect a higher success ratio under tight deadline conditions for PCP_v01, the greedy version with a finishing repair cycle, than for values above 2.5.

A first trial to test this conjecture is by generating random task graphs by means of the open source random graph generator GGen [40]. Two sets are generated for task graphs having size 32 and 64 vertices where the in-degree parameter had values between 2 and 5 and the out-degree parameter values between 1 and 4. This choice prevents the GGen algorithm from generating simple task graphs, for instance, task graphs with a single path. For each size of a task graph the generated graph was checked for the number of paths per vertex, for a value less than 2.5 the graph was added to a set called GG 32_0/GG 64_0, otherwise, it was added to a set called GG 32_1/GG 64_1. Once each set contained 20 examples, for each example computation times and execution times were generated for the different VMs following the same procedure as explained above.

Fig. 7 displays the success ratio of the PCP_v01 algorithm for the different sets, and for both sizes of task graphs, the success rate is larger if the number of paths per task is less than 2.5. A first tentative result to test a conjecture based upon the topology of a task graph.

The metric applied is a topological metric of the task graph that should be distinguished from a more detailed metric which quantifies the granularity of a DAG as defined by Gerasoulis and Yang [41] and applied by McCreary et al. [33]. They take the communication cost and the execution cost into account, whereas a workflow DAG for the cloud only defines the dependencies of the tasks and the communication cost among tasks and not the execution times of the tasks. These result from the outcome of the scheduling algorithm, although their metric might be applied to the start configuration with each task assigned to the fastest VM.

5. Discussion and future work

In this paper, we profiled two implementations of the IC-PCP algorithm with different rules of greediness for assigning the tasks of a partial critical path to a virtual machine. Through the systematic profiling proposed in this paper, the difference in performance of both implementations of the IC-PCP algorithm could be made more explicit as a function of different types of workflows and QoS requirements, this for the Real-World Scientific Workflows from the Pegasus project [22]. Besides the

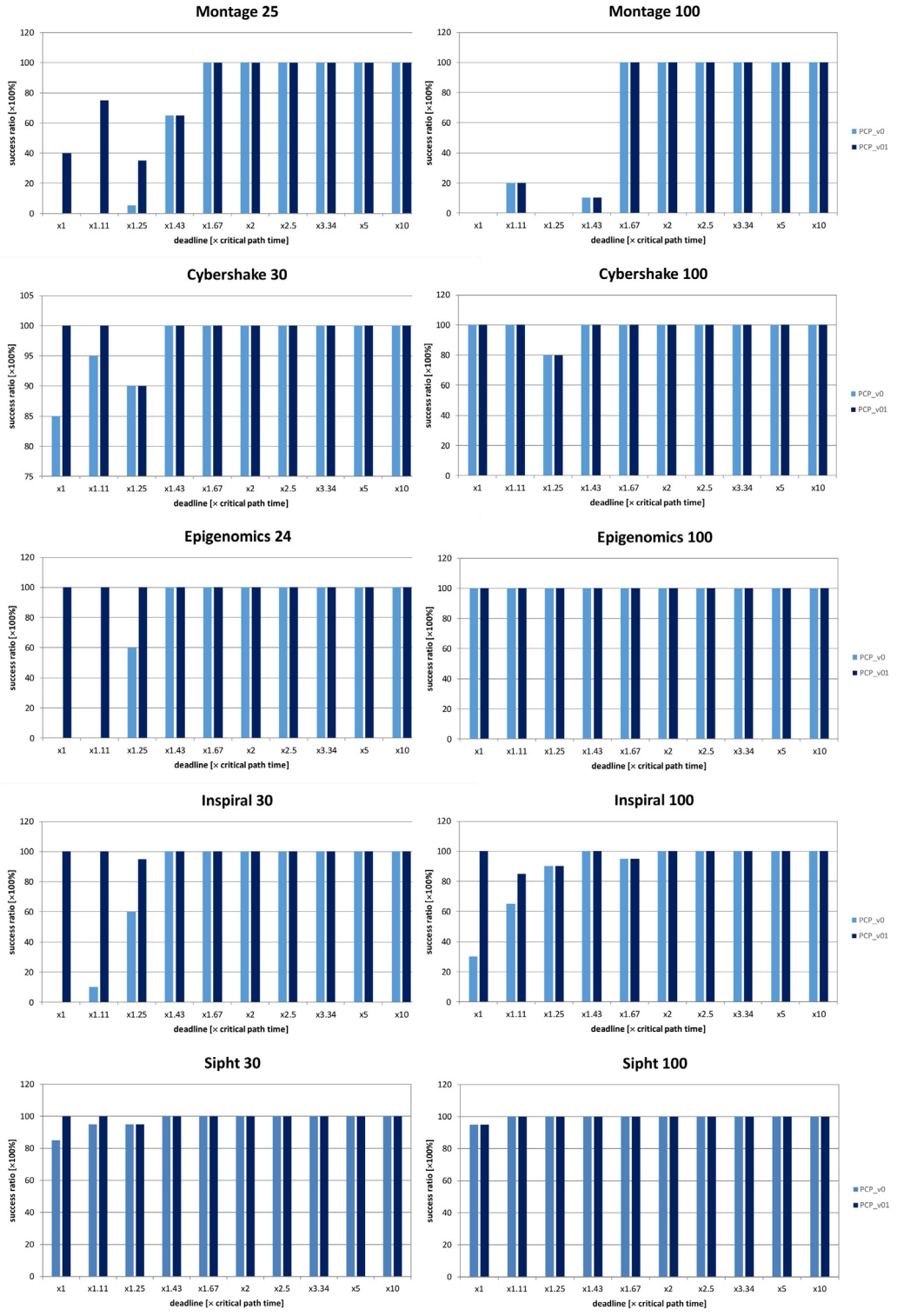


Fig. 5. The improvement of the success ratio of PCP_v0 by PCP_v01 with a finishing repair cycle, for the different topology sets. The values $\{\times 1, \times 1.11, \ldots, \times 10\}$ for the deadline correspond with the values $\{100, 90, \ldots, 10\}$ for the stress parameter p.

greediness of the heuristics, the deadlines and the workflow type are important features. The closer the deadline of the workflow problem to the time of the critical path for the fastest VMs, different success rates in solving a specific workflow problem became apparent for the IC-PCP algorithm. A striking feature is the fact that the greedy implementation of the IC-PCP algorithm could be enhanced. In case the greedy implementation failed to find a solution under tight deadline conditions and ended in an invalid not implementable state, a finishing repair cycle, (Fig. 5), was able to turn the invalid final state into a valid solution.

However, if the input workflow is a workflow 'similar' to Montage (Figs. 3 and 5), we observed that a repair cycle did not improve the success rate in solving the workflow problem. Another feature that becomes apparent from Fig. 6 is that the less greedy, more stringent, version produces solutions with the same mean cost as the greedy version under less tight deadline

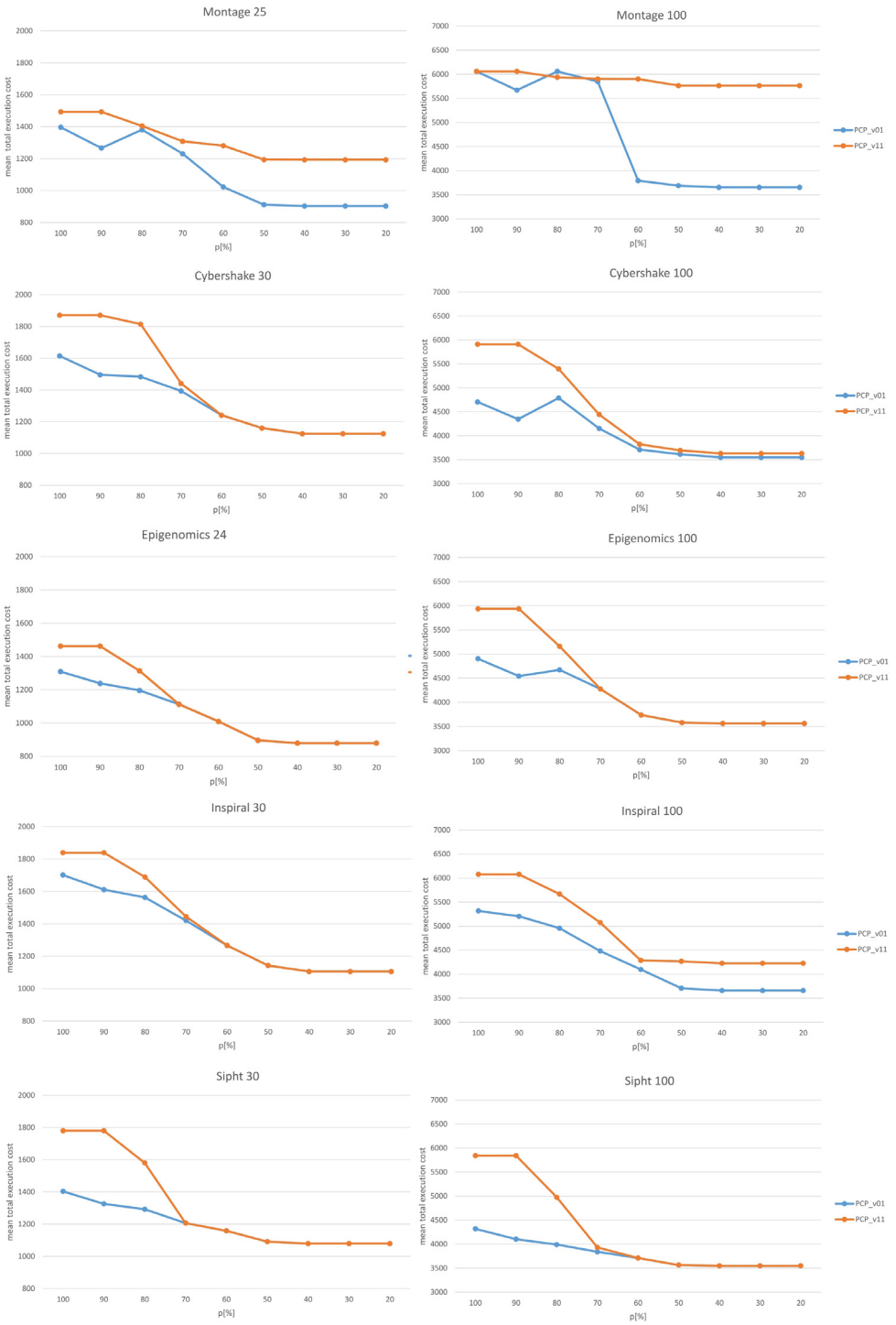


Fig. 6. The mean total execution cost of the enhanced greedy algorithm PCP_v01 and the enhanced more stringent algorithm PCP_v11 for different topology sets, each set having 20 DAGs. The values $\{\times 1, \times 1.11, \ldots, \times 10\}$ for the deadline correspond with the values $\{100, 90, \ldots, 10\}$ for the stress parameter p. In case an algorithm fails to find a solution, the cost equals the cost of the start configuration with VMs of the most expensive type offered.

conditions except for the Montage topologies. This asked for a way to characterize the Montage workflow problem. We tried this by applying a graph metric for the DAG of the workflow problem that could discriminate between the Montage workflows and the other Real-World workflow problems. The graph metric was tested by generating two random sets of workflow problems (DAGs), one set with workflow problems 'similar' to the Montage workflow and the other set with workflow problems more 'similar' to the other Real-World workflow problems.

As we think that every scheduling approach based on heuristics or without proof of convergence for an NP-hard problem will fail for a certain kind of scheduling problem under tight deadline conditions, we propose to add the different steps of IC-PCP profiling presented above into a profiling framework Fig. 2. By applying such a profiling framework, one can recommend heuristics and greediness for a given scheduling algorithm given workflows with different characteristics. If an impairment of performance of a scheduling approach is discovered for a certain kind of workflow problems, it is a challenge to find a graph metric or a topological

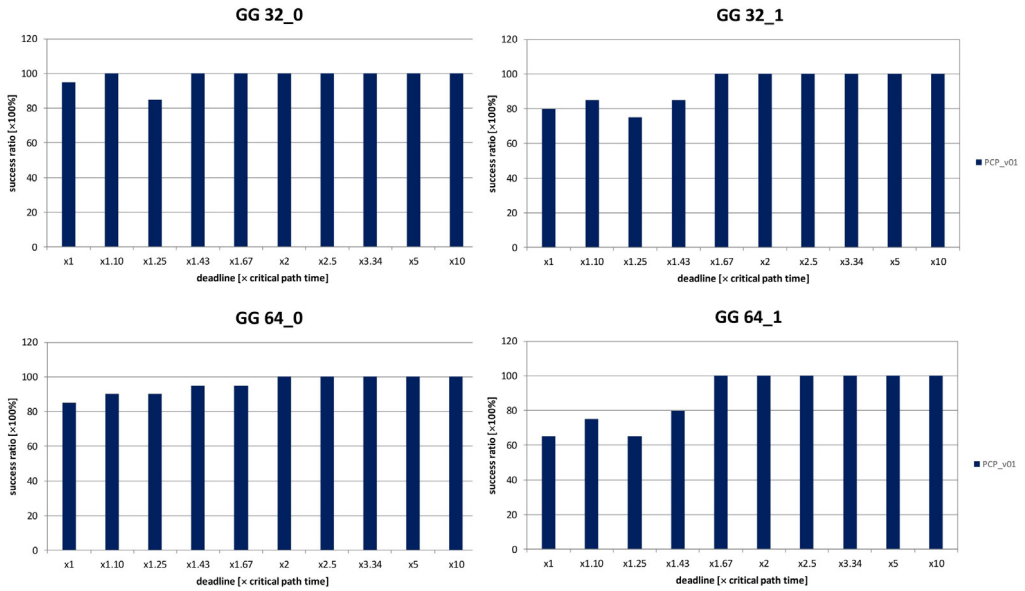


Fig. 7. The success ratio of PCP_v01, the greedy version with a finishing repair cycle, for different topology sets, each set having 20 DAGs. DAGs from the sets GG 32_0 and GG 64_0 have a value for $\#paths/|V|$ below 2.5, whereas DAGs from the sets GG 32_1 and GG 64_1 have a value above 2.5. The values $\{\times 1, \times 1.11, \ldots, \times 10\}$ for the deadline correspond with the values$\{100, 90, \ldots, 10\}$ for the stress parameter p.

index that can characterize these difficult workflow problems. Difficulty in finding a graph metric is the fact that not all metrics have a meaning for directed acyclic graphs. More research in this field might be facilitated by looking at the use of topological indices in other fields of science, e.g. molecular sciences [42]. During the search for a 'successful' graph metric, we noticed that metrics that are purely degree based were in general not able to discriminate between the topology of Montage DAG and the DAGs of other Real-World workflow problems. Therefore, we think that more success can be expected from metrics that take the number of different paths in the DAG into account.

From a practical point of view, one might run different scheduling approaches in parallel in solving a single workflow problem and take the result of the winner. However, from a theoretical perspective, it is still of interest to characterize the success ratio of an algorithmic approach with a certain heuristic, or greediness, and to study its behavior under different tight deadline conditions and or topology features of the task graph.

A closer look at Fig. 3 raises the question of what makes topologies like Cybershake and Sipht rather insensitive to tight deadline conditions for the greedy approach. Compared to the other topologies the success rate reaches almost 100% for the whole range of the stress parameter. Another interesting question is under which circumstances a greedy implementation might be susceptible for a finishing repair cycle to improve its success rate under tight deadline conditions as found for the IC-PCP algorithm.

The proposed profiling procedure might also be suitable to study scheduling algorithms applying meta-heuristics including genetic algorithms (GA), particle swarm optimization (PSO) and ant colony optimization algorithms (ACO). From our experience, we know that these approaches become very time consuming compared to e.g. the IC-PCP algorithm, when the size of the task graph (DAG) becomes large.

To the best of our knowledge, this is the first attempt to build a systematic profiling framework suited to study different static scheduling algorithms.

Code and data availability

All four implementations of the IC-PCP algorithm used in this study as well as the data to produce the performance figures are available at https://bitbucket.org/uva-sne/ic-pcp-profiling/src/master.

Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIPLUS project), 676247 (VRE4EIC project), 824068 (ENVRI-FAIR project) and 825134(ARTICONF project).

Declaration of competing interest

Arie Taal, Junchao Wang, Cees de Laat, and Zhiming Zhao declare that they have no conflict of interest.

References

[1] M. Mizielinski, M. Roberts, P. Vidale, R. Schiemann, M.-E. Demory, J. Strachan, T. Edwards, A. Stephens, B. Lawrence, M. Pritchard, et al., High-resolution global climate modelling: The upscale project, a large-simulation campaign, Geosci. Model Dev. 7 (4) (2014) 1629–1640.

[2] S. Fang, L. Xu, Y. Zhu, Y. Liu, Z. Liu, H. Pei, J. Yan, H. Zhang, An integrated information system for snowmelt flood early-warning based on internet of things, Inf. Syst. Front. 17 (2) (2015) 321–335.

[3] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, R. Ranjan, Osmotic flow: Osmotic computing + iot workflow, IEEE Cloud Comput. 4 (2) (2017) 68–75.

[4] Z. Zhao, A. Taal, A. Jones, I. Taylor, V. Stankovski, I.G. Vega, F.J. Hidalgo, G. Suciu, A. Ulisses, P. Ferreira, C.d. Laat, A software workbench for interactive, time critical and highly self-adaptive cloud applications (switch), in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 1181–1184 http://dx.doi.org/10.1109/CCGrid.2015.73.

[5] T. Lorido-Botran, J. Miguel-Alonso, J.A. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, J. Grid Comput. 12 (4) (2014) 559–592.

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

[7] C. Jiang, C. Wang, X. Liu, Y. Zhao, A survey of job scheduling in grids, in: Proceedings of the Joint 9th Asia-Pacific Web and 8th International Conference on Web-Age Information Management Conference on Advances in Data and Web Management, ACM, 2007, pp. 419–427.

[8] N. Nicholas Taleb, The black swan: The impact of the highly improbable, Victoria 250 (2015) 595–7955.

[9] F. Fakhfakh, H.H. Kacem, A.H. Kacem, Workflow scheduling in cloud computing: A survey, in: IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, IEEE, 2014, pp. 858–865, http://dx.doi.org/10.1109/EDOCW.2014.61.
[10] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.
[11] S. Smanchat, K. Viriyapant, Taxonomies of workflow scheduling problem and techniques in the cloud, Future Gener. Comput. Syst. 52 (2015) 1–12.
[12] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: A survey, J. Supercomput. 71 (2015) 3373–3418.
[13] E.N. Alkhanak, S.P. Lee, R. Rezaei, R.M. Parizi, Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues, J. Syst. Softw. 113 (2016) 1–26.
[14] W. Zheng, R. Sakellariou, Budget-deadline constrained workflow planning for admission control, J. Grid Comput. 11 (2013) 633–651.
[15] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[16] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694.
[17] J. Shi, J. Luo, F. Dong, J. Zhang, J. Zhang, Elastic resource provisioning for scientific workflow scheduling in cloud under budget and deadline constraints, Cluster Comput. 19 (1) (2016) 167–182, http://dx.doi.org/10.1007/s10586-015-0530-0.
[18] Z. Zhu, G. Zhang, M. Li, X. Liu, Evolutionary multi-objective workflow scheduling in cloud, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1344–1357, http://dx.doi.org/10.1109/TPDS.2015.2446459.
[19] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, Z. Zhao, Planning virtual infrastructures for time critical applications with multiple deadline constraints, Future Gener. Comput. Syst. 75 (2017) 365–375, http://dx.doi.org/10.1016/j.future.2017.02.001, URL: http://www.sciencedirect.com/science/article/pii/S0167739X17301905.
[20] S. Pandey, L. Wu, S.M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: Proceedings of 2010 24th IEEE International Conference on Advanced Information Networking and Applications, AINA, IEEE, 2010, pp. 400–407, http://dx.doi.org/10.1109/AINA.2010.31.
[21] S. Xue, M. Li, X. Xu, J. Chen, An aco-lb algorithm for task scheduling in the cloud environment, J. Softw. 9 (2) (2014) 466–473.
[22] Workflow generator. URL: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[23] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2013) 682–692.
[24] Y.C. Lee, H. Han, A.Y. Zomaya, M. Yousif, Resource-efficient workflow scheduling in clouds, Knowl.-Based Syst. 80 (2015) 153–162.
[25] A. Ilyushkin, D. Epema, The impact of task runtime estimate accuracy on scheduling workloads of workflows, in: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2018.
[26] Z. Zhao, P. Grosso, J. van der Ham, R. Koning, C. de Laat, An agent based network resource planner for workflow applications, Int. J. Multi-Agent Grid Syst. 7 (6) (2011) 187–202.
[27] K. Jeferry, G. Kousiouris, D. Kyriazis, J. Altman, A. Ciuffoletti, I. Maglogiannis, P. Nesi, B. Suzic, Z. Zhao, Challenges emerging from future cloud application scenarios, Procedia Comput. Sci. 68 (2015) 227–237, http://dx.doi.org/10.1016/j.procs.2015.09.238.
[28] Y. Hu, H. Zhou, C. de Laat, Z. Zhao, Ecsched: Efficient container scheduling on heterogeneous clusters, in: Proceedings of the Euro-Par 2018 Conference, 2018 http://dx.doi.org/10.1007/978-3-319-96983-1_26.
[29] H. Zhou, Y. Hu, J. Su, C. de Laat, Z. Zhao, Empowering dynamic task-based applications with agile virtual infrastructure programmability, in: Proceedings of the IEEE International conference on Cloud, San Francisco, USA, 2018 http://dx.doi.org/10.1109/CLOUD.2018.00068.
[30] J. Kelley, M. Walker, Critical-path planning and scheduling, in: Proceedings of the Eastern Joint Computer Conference. New York, USA, 1959.
[31] J.E. Kelley, Critical-path planning and scheduling: Mathematical basis, Oper. Res. 9 (3) (1961) 167–179.
[32] J. Zhou, P. Love, X. Wang, K. Teo, Z. Irani, A review of methods and algorithms for optimizing construction scheduling, J. Oper. Res. Soc. 64 (2013) 1091–1105.
[33] C. McCreary, M. Cleveland, A. Khan, The problem with critical path scheduling algorithms, Technical Report, 9601.
[34] S.A. Curtis, The classification of greedy algorithms, Sci. Comput. Program. 49 (1–3) (2003) 125–157.
[35] G.B. Berriman, E. Deelman, J.C. Good, J.C. Jacob, D.S. Katz, C. Kesselman, A.C. Laity, T.A. Prince, G. Singh, M.-H. Su, Montage: A grid-enabled engine for delivering custom science-grade mosaics on demand, in: Optimizing Scientific Return for Astronomy through Information Technologies, vol. 5493, International Society for Optics and Photonics, 2004, pp. 221–233.
[36] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, et al., Scec cybershake workflows—automating probabilistic seismic hazard analysis calculations, in: Workflows for E-Science, Springer, 2007, pp. 143–163.
[37] Usc epigenome center. URL: http://epigenome.usc.edu.
[38] Ligo project. URL: http://www.ligo.caltech.edu/.
[39] Compute benchmark scores. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/compute-benchmark-scores.
[40] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, F. Wagner, Random graph generation for scheduling simulations, in: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, p. 60.
[41] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, IEEE Trans. Parallel Distrib. Syst. 4 (6) (1993) 686–701.
[42] Q.-N. Hu, Y.-Z. Liang, K.-T. Fang, The matrix expression topological index and atomic attribute of molecular topological structure, J. Data Sci. (2003) 361–389.

Arie Taal is currently a researcher at the University of Amsterdam. His research focuses on advanced networks and cloud computing.

Junchao Wang is currently a Ph.D. student in University of Amsterdam. His research interests focus on cloud computing and Software-Defined Networking technologies and how to apply these technologies to guarantee the system level QoS of time critical applications.

Prof. Dr. Ir. Cees de Laat is chair of the System and Network Engineering research group at the University of Amsterdam. Research in his group includes optical/switched Internet for data-transport in TeraScale eScience applications, Semantic web to describe networks and associated resources, distributed cross organization Authorization architectures and Systems Security privacy of information in distributed environments. He serves as at Large member of the Board of Directors in Open Grid Forum and is acting co-chair of the Grid High Performance Networking Research Group (GHPN-RG), is chair of GridForum.nl and boardmember of ISOC.nl. He is co-founder and organizer of several of the past meetings of the Global Lambda Integrated Facility (GLIF) and founding member of http://CineGrid.org.

Zhiming Zhao obtained his Ph.D. in computer science in 2004 from University of Amsterdam (UvA). He is currently a senior researcher in the System and Network Engineering group at UvA. He coordinates research effort on quality critical systems on programmable infrastructures in the context of European H2020 projects of SWTICH, ENVRIPLUS and VRE4EIC. His research interests include software defined networking, workflow management systems, multi agent system and big data management.