Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Pattern learning for scheduling microservice workflow to cloud containers

Wenzheng Li
Southeast University
Xiaoping Li
xpli@seu.edu.cn
Southeast University
Long Chen
Southeast University

Research Article

Keywords: Service computing, Scheduling optimization, Microservice workflow, Container cloud, Pattern recognition

Posted Date: December 13th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3726089/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at International Journal of Machine Learning and Cybernetics on March 16th, 2024. See the published version at https://doi.org/10.1007/s13042-024-02115-5.

Pattern learning for scheduling microservice workflow to cloud containers

Wenzheng Li[1], Xiaoping Li[1*], Long Chen[1]

[1*]School of Computer Science and Engineering, Southeast University, Nanjing, 211189, Jiangsu, China.

*Corresponding author(s). E-mail(s): xpli@seu.edu.cn;
Contributing authors: vinson@seu.edu.cn; longc@seu.edu.cn;

Abstract

Patterns are crucial for efficiently scheduling microservice workflow applications to containers in cloud computing scenarios. However, it is challenging to learn patterns of microservice workflows because of their complex precedence constrained structures provided by users with more lightweighted, diversified, and personalized services. In this paper, we propose a graph neural network is designed to identify patterns within a set of microservice workflows by mining the common substructures of workflows. Based on the learned patterns, a pattern-based scheduling algorithm framework is developed for microservice workflows with soft deadline constraints to minimize the average tardiness. A sorting strategy is introduced based on urgency and pattern coverage rate. For simplification of the task sorting process, the pattern-based task sorting algorithm (PB-TS) is devised. Furthermore, a resource selection phase is incorporated to the pattern-based resource selection algorithm (PB-RS) to minimize the candidate resource space. Experimental results demonstrate the proposed method is much efficient as compared to three classical algorithms.

Keywords: Service computing, Scheduling optimization, Microservice workflow, Container cloud, Pattern recognition

1 Introduction

More and more commercial enterprises, e.g., Amazon and Netflix [1, 2], adopt the microservice architecture for diverse and personalized complex service requirements. These requirements are usually microservice workflow applications [3] which are depicted by DAGs (directed acyclic graphs). Compared to general cloud workflow, tasks of a microservice workflow are characterized by larger-scale, smaller-granularity and more complex dependent relationships [4]. Generally, they are constrained by different soft deadlines. These characteristics make the problem of scheduling microservice workflows much more complicated than that of general cloud workflows, especially much more time-consuming. However, scheduling microservice workflows in a short time is necessary or even demanded in many application scenarios, e.g., traffic monitoring, securities trading. In fact, a lot of computations are repeated in a scheduling process because there are a lot of similar or same substructures in the microservice workflows [5], which are called patterns in this paper. The scheduling efficiency could be improved significantly if the patterns are fulfilled by composed services.

In this paper, we consider the problem of scheduling a set of microservice workflows to heterogeneous containers in cloud computing [6, 7], submitted by different customers. The considered problem is different from the classic VM (virtual machine)-based workflow scheduling problem [8]. Microservice workflows are constrained by soft deadlines and contain many patterns. Tasks in each workflow are precedence constrained. Containers have different processing speeds for different types of microservices. It is desirable to meet the soft deadline requirements of all customers, so the average tardiness is optimized in the considered problem.

Because of the large-scale and smaller-granularity tasks with much more complex dependent relationships, it is rather difficult to schedule microservice workflows efficiently based on the following main challenges:

- Because of the complex structures of microservice workflows, it is challenging to recognize the patterns quickly and accurately.
- Different soft deadlines are required by customers. It is crucial and difficult to get an appropriate workflow sequence to minimize the average tardiness.
- The large number of tasks in each microservice workflow have a great influence on the optimization objective. How to sort the tasks of each microservice workflow is also challenging.
- It is much more difficult to allocate massive heterogeneous containers to tasks with both pattern and non-pattern features.

For the above challenges of the considered problem, the PMWSC (Pattern-based Microservice Workflow Scheduling to Containers) framework is proposed which consists of four components: pattern recognition, workflow sequencing, task sorting, and resource selection. The main contributions are as follows:

- Based on graph neural network, a pattern recognizer is designed to mine patterns in a set of microservice workflows. Semi-supervised learning is adopted to achieve patterns fast and accurately.
- A workflow sequencing strategy is proposed for a set of microservice workflows in terms of the priorities defined by soft deadlines, critical paths and pattern coverage rates.
- A pattern-based task sorting algorithm (PB-TS) is developed which reduces the computation time by reusing the ordered tasks of the patterns, i.e., a lot of repeated calculations are avoided.
- A pattern-based resource selection algorithm (PB-RS) is proposed which utilizes historical selecting schemes to reduce resource search time.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. Section 3 modifies the system architecture and constructs the corresponding mathematical model. The proposed methods for the considered problem are described in Section 4. Section 5 evaluates the performance of the proposed methods followed by conclusions in Section 6.

2 Related Work

In order to design efficient scheduling algorithms for microservice workflow, both general workflow and microservice workflow scheduling problems are well studied. In the study of general workflow scheduling, we focus on the analysis of the advantages and disadvantages of various workflow scheduling algorithms. In the research of microservice workflow scheduling, we focus on the implementation of the current microservice workflow scheduling architecture.

2.1 General Workflow Scheduling

Workflow scheduling is one of the most important problems in cloud computing. It is usually NP-complete, i.e., it is almost impossible to find the optimal solution in polynomial time. Many algorithms have been proposed which can be divided into three categories: heuristic, meta-heuristic and learning-based algorithm.

Heuristics include list scheduling (such as HEFT [9] and PEFT [10]), cluster-based scheduling [11] and task replication scheduling [12]. List scheduling usually divides the scheduling process into two stages: task sequencing and resource selection. Different heuristic rules could be embedded in any of the two stages. For example, many sub-deadline division strategies were proposed [13] to meet the deadline constraint, such as the

deadline-Markov decision process [14], the deadline early tree [15], the critical path-based iteration [16] and partial critical paths [17]. Abrishami et al. [18] defined Partial Critical Paths (PCPs) and proposed the IC-PCP algorithm for deadline constrained workflow scheduling. Wu et al. [3] extended the upward rank used in HEFT and defined a probabilistic upward rank to meet the deadline-constraint. The cluster-based scheduling algorithm proposed in [11] allocates different task clusters to specific resources through the proposed allocation policies. Different from cluster-based scheduling, we call the similar substructures in a workflow as a pattern rather than a cluster in this paper. A pattern is a subgraph with a node set and an edge set, which is different from a cluster of tasks. Task replication scheduling [12] copies tasks to reduce communication costs and minimize the completion time of all tasks.

Typical meta-heuristic algorithms are PSO (Particle Swarm Optimization) [19], ACO (Ant Colony Optimization) [20], GA (genetic algorithm) [21], and SA (Simulated Annealing) [22]. Though a meta-heuristic method is usually more effective for finding better scheduling solutions than a heuristic, it is more time-consuming. Therefore, meta-heuristics are not suitable for cloud application scheduling problems with rapid response requirements.

The efficiency of scheduling algorithms for complex workflows is rather important in practice. Knowledge (e.g., expert experience) obtained by machine learning is utilized to improve scheduling algorithm efficiency. Melnik et al. [23] proposed a scheduling algorithm based on a neural network to minimize the completion time of workflows. Similarly, Kintsakis et al. [24] used machine learning to predict the running time of workflow tasks and the failure probability of specific tasks assigned to computing resources. In order to optimize the response time, some studies applied reinforcement learning to workflow scheduling. Cui et al. [25] proposed a reinforcement learning-based method for workflow scheduling with multiple priorities submitted to clouds. Wu et al. [26] designed an improved Q-learning algorithm with a weighted fitness value function to optimize makespan and balance loads. However, the effectiveness of learning-based methods depends on the dataset with rich label information, which is not suitable for complex applications with uncertain distribution. Many learning-based scheduling algorithms directly use neural networks as end-to-end schedulers to obtain scheduling schemes. These methods do not extract pattern structures of intermediate processes and the obtained results are usually non-interpretable. Recently, GNN (graph neural network) has been commonly used for graph pattern recognition [27], which can support a variety of graph analyses, e.g., link prediction, node classification, and community structure recognition [28–32]. Using flexible representation learning, these GNN methods can find more accurate patterns [33]. GNN has been used to solve the problem of graph matching and graph similarity learning [34, 35]. However, only a few studies applied GNN to the cloud workflow scheduling problem.

2.2 Microservice Workflow Scheduling

Besides the traditional workflow scheduling, more and more attention has been paid on microservice workflow scheduling with provisioned container resources [36–38]. These related studies mainly focus on the system architecture of microservice workflows in practice. A distributed service workflow engine based on microservices was developed by Kurhinen et al. [4] based on the service-oriented architecture framework. Some open issues in the microservices scheduling and resources management were summarized by Fazio et al. [39], e.g., how to deploy microservices to containers using some tools (such as Docker Swarm, Kubernetes) to instantiate and how to manage containers in clouds. Zheng et al. [40] introduced a container sharing mechanism to workflow tasks to eliminate notable overheads caused by large workloads. The Skyport proposed in [41, 42] is a container-based execution management system for multi-cloud scientific workflows. Using containers to existing scientific workflow platforms, the software could be well deployed and resource utilization could be improved. Bhamare et al. [43] addressed the problem of scheduling microservices across multiple clouds with different user-level SLAs, e.g.,latency and cost. By evaluating the performance of the microservices architecture using containers, Amaral et al. [44] demonstrated that containers are

suitable for microservices because of the characteristics of lightweight, fast start-up times, and low overheads.

Resource allocation and resource provisioning are key components in cloud workflow scheduling. Most microservices are performed in container environments. Containers are incorporated into many existing cloud computing platforms to automatically deploy on the provided IaaS, e.g., Amazon EC2 Container Service, Google Container Engine, Microsoft Azure Container Service and IBM Bluemix. At present, most cloud service providers provide container as a service (CaaS). Based on CaaS, Tihfon et al. [45] proposed a multi-task cloud infrastructure which is flexible and efficient for optimizing application performance using the proposed scheduling and load balancing algorithms. Peinl et al. [46] surveyed the state of the art on the VM and container management tools in cloud computing ecosystems.

To the best of our knowledge, containers have been widely used to microservice applications. It is much more time-consuming to schedule a large number of microservice tasks to containers, but there are many similar substructures in microservice applications, and it is possible to reduce the computation time of the scheduling algorithm. However, little attention has been paid on mining similar substructures to speed up scheduling algorithms. In this paper, we focus on graph pattern recognition to find similar substructures of microservice workflows based on which an efficient scheduling algorithm is developed.

3 System Architecture and Problem Formulation

3.1 System Architecture

Based on the structure given in [47], a modified architecture is developed as shown in Figure 1 which contains three components: the user interface, Microservice Workflow Management System (MWMS), and the container cloud with Container as a Service (CaaS). Similarly to the architecture in [48], the Controller of MWMS further contains the Cloud Manager, the Container Allocator, and the Monitor. Container resources are dynamically managed by Controller. All container resources are assumed to be in the same resource space pool without considering in which hosts or virtual machines they are located.

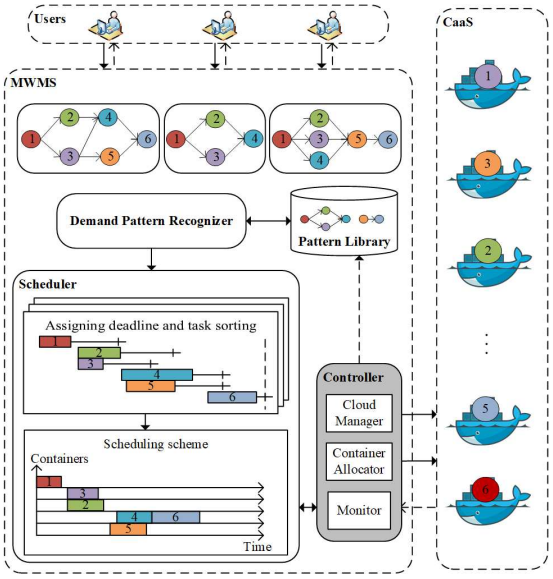


Fig. 1 Modified architecture for microservices workflows.

The workflows submitted by different users are scheduled by Scheduler after the pattern recognition. Once the patterned tasks are sorted and the required resources are selected, the obtained topological sequence and the task-resource mappings are recorded in the pattern library which could be reused in the following scheduling process.

Assume that MWMS accepts n microservice applications $W = \{W_1, W_2, \ldots, W_n\}$. Application W_i with μ_i nodes is described by a DAG $W_i = (V_i, E_i)$. $V_i = \{v_{i,1}, \ldots, v_{i,\mu_i}\}$ represents the microservice task set of W_i. $E_i = \{(v_{i,j'}, v_{i,j}) | v_{i,j'}, v_{i,j} \in V_i\}$ is the edge set between microservice tasks. $(v_{i,j'}, v_{i,j})$ means that $v_{i,j}$ cannot start until task $v_{i,j'}$ completes. $\varphi^P_{i,j}$ and $\varphi^s_{i,j}$ are the set of immediate predecessors and the set of immediate successors of $v_{i,j}$, respectively. The execution time of task $v_{i,j}$ is $T^e_{i,j}$.

There are a large number of containers in CaaS. The container set is denoted as $C = \{c_1, c_2, \ldots, c_q\}$ in which $q \gg \sum_{i=1}^{n} u_i$. Different from virtual machines, containers can be configured more flexibly which leads to a much larger number and more heterogeneous states than VMs. These characteristics result in that it is much more difficult to find the most appropriate containers for the concerned microservice tasks than

to find appropriate VMs. Therefore, the problem under study is much more complex than that of scheduling workflow applications to VMs. Since the problem of scheduling workflow applications to VMs is NP-hard, it is natural that the considered problem is NP-hard.

3.2 Scheduling Problem Description

For the considered scheduling problem, we make the following assumptions:

- The scheduler accepts a large number of workflows with similar substructures.
- Microservice tasks of the workflows are non-interruptible and non-divisible. Each microservice task is executed on only one container at a time.
- Same to most workflow scheduling algorithms [18, 49–52], each cloud resource performs only one task at a time.
- All containers are in the same resource pool and their startup times are ignored.

Notations to be used are listed in Table 1.

Table 1 Notations to be used.

n	Number of microservice workflows
W	Set of microservice workflows, $W = \{W_1, W_2, \ldots, W_n\}$
W_i	The i^{th} microservice workflow in W
μ_i	Number of tasks in microservice workflow W_i
μ_i^p	Number of tasks with pattern feature in microservice workflow W_i
V_i	Set of tasks in W_i, $V_i = \{v_{i,1}, \ldots, v_{i,\mu_i}\}$
$v_{i,j}$	The j^{th} task in microservice workflow W_i
E_i	Set of precedence relationship from task $v_{i,j}$ to $v_{i,j'}$, $E_i = \{(v_{i,j'}, v_{i,j'}) \mid v_{i,j'} \in V, v_{i,j} \in V_i\}$
$\varphi_{i,j}^P$	Immediate predecessor set of $v_{i,j}$
$\varphi_{i,j}^s$	Immediate successor set of $v_{i,j}$
A_i	Arrival time of W_i
D_i	Soft deadline of W_i
MS	Set of microservice function components
$ms_{i,j}$	Microservice function of the j^{th} task $v_{i,j}$ $ms_{i,j} \in MS, MS = \{ms_1, ms_2, \ldots, ms_p\}$
q	Number of container resources
C	Set of containers $C = \{c_1, \ldots, c_q\}$
c_k	The k^{th} container in C
s_{k,ms_j}	Speed of the k^{th} container for microservice ms_j
P	Set of workflow pattern in the pattern library, $P = \{P_1, P_2, \ldots, P_n\}$
x_v	Input vector of task v in GNN
$c_{v,u}$	Input vector of edge in GNN, u is the immediate successor of v
e_v	Output feature of task v in GNN

A scheduling scheme is described by $\pi = \{m_{i,j,k} | m_{i,j,k} = (v_{i,j}, c_k, ST(v_{i,j}, c_k))\}$ in which $m_{i,j,k}$ represents that task $v_{i,j}$ of workflow W_i is scheduled to container c_k with the start time $ST(v_{i,j}, c_k)$. The corresponding execution time and completion time are calculated by:

$$ET(v_{i,j}, c_k) = \frac{w_j}{s_{k,ms_j}} \tag{1}$$

$$FT(v_{i,j}, c_k) = ST(v_{i,j}, c_k) + ET(v_{i,j}, c_k) \tag{2}$$

The start and finish times $LST(c_k)$ and $LFT(c_k)$ of all tasks in c_k are determined by:

$$LST(c_k) = \min_{v_{i,j} \in Sche(c_k)} \{ST(v_{i,j}, c_k)\} \tag{3}$$

$$LFT(c_k) = \max_{v_{i,j} \in Sche(c_k)} \{FT(v_{i,j}, c_k)\} \tag{4}$$

where $Sche(c_k)$ is the task set scheduled to c_k.

Assume tasks $v_{i,j}$ and $v_{i,p}$ are scheduled to containers c_x and c_y, respectively. The bandwidth and latency of the two containers are $b_{x,y}$ and $d_{x,y}$. $Avail(c_k)$ is the earliest available time of c_k determined by

$$Avail(c_k) = \max_{v_{i',j'} \in Sche(c_k)} \{AFT(v_{i',j'})\} \tag{5}$$

where $v_{i',j'}$ is the j'^{th} task of $W_{i'}$ existing in c_k before accepting $v_{i,j}$. The start time $ST(v_{i,j}, c_k)$ of $v_{i,j}$ satisfies $ST(v_{i,j}, c_k) \geqslant \max\{Avail(c_k), \max_{p \in \varphi_j^P}\{AFT(v_{i,p}) + TT(v_{i,p}, v_{i,j})\}\}$ in which $TT(v_{i,p}, v_{i,j})$ is the data transmission time from $v_{i,p}$ to $v_{i,j}$ determined by $TT(v_{i,p}, v_{i,j}) = \frac{data_{p,j}}{b_{x,y}} + d_{x,y}$. The response time RT_i and tardiness TD_i of workflow W_i can be calculated as follows:

$$RT_i = \max_{v_{i,j} \in W_i} \{AFT(v_{i,j})\} - A_i \tag{6}$$

$$TD_i = \max\{RT_i - D_i, 0\} \tag{7}$$

where D_i is the soft deadline of W_i. The objective of the problem under study is to minimize the average tardiness $\overline{TD}$:

$$\min\ \overline{TD} = \frac{\sum_{i \in W} TD_i}{n} \tag{8}$$

4 Proposed Methods

For the considered problem, the PMWSC (Pattern-based Microservice Workflow Scheduling to Containers) framework is proposed as shown in Figure 2. A set of workflows submitted by different users is received by the scheduling system and their similar structures are recognized. Tasks in the pattern features are labeled. A workflow scheduling sequence is determined according to the soft deadline and the rate of covered patterns. Tasks of each workflow are scheduled in terms of the priorities determined by the topological sequence. Appropriate resources are selected for all of the tasks meeting the constraints.

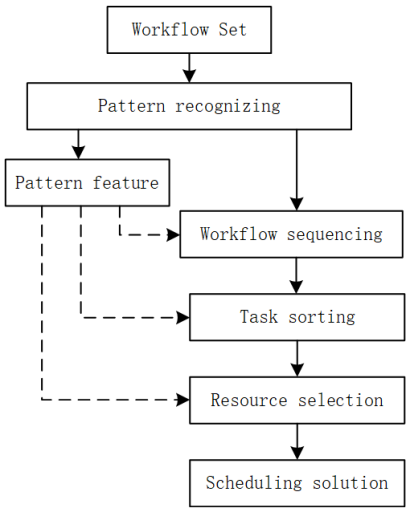


Fig. 2 The proposed PMWSC framework.

The scheduling algorithm process (as shown in Algorithm 1) consists of four phases: pattern recognizing, workflow sequencing, task sorting, and resource selection. The GNN (graph neural network)-based workflow pattern recognizer analyzes the structure of a workflow and outputs the pattern feature of its tasks (line 1). Based on the urgency and pattern coverage rate, the scheduling order of the workflow is obtained (line 2). Pattern features are used to speed up the task sorting stage and the resource selection stage of the scheduling process (line 4,5). Details of each algorithmic component are described in the following.

Algorithm 1: Pattern-based Microservice Workflow Scheduling to Containers (PMWSC)

1 Input the set of workflow application W to the pattern recognizer and get the pattern feature of each task $v_{i,j}$;
2 Generate workflow sequence $W' = (W_{[1]}, \ldots, W_{[n]})$ by a workflow sequencing strategy;
3 **for** each $W_{[i]}$ in W' **do**
4 Call PB-TS;
5 Call PB-RS;

4.1 Pattern Recognizing

Since patterns are crucial for the performance of scheduling algorithms, it is desirable to recognize patterns accurately and fast. Patterns could be recognized either on-line or off-line. It is important to recognize patterns in advance for workflows with similar sub-structures. Because of the advantage of GNN encoding node information (e.g., the running time of tasks denoted by nodes, the dependency structure between nodes, the communication time between tasks, and the state of processing units) into a set of embedding vectors using node embedding learning, a GNN-based offline pattern recognizer is designed for large-scale workflows in this paper.

As shown in Fig. 3, DAGs of workflows are input to the GNN model through which the information of each node (including its running time, its adjacent nodes, communication times of the connected edges) is learned. By learning historical data in this way, a task pattern feature library is obtained. The learned node information also can be searched from the pattern feature library. If there is a record in the library, the task is labeled by a pattern task. The pattern library includes a set of workflow patterns $P = \{P_1, P_2, \ldots, P_n\}$. Let e_v be the pattern feature of task v. $e_{v_i} = e_{v_j}$ if and only if tasks v_i and v_j have the same pattern feature, i.e., tasks v_i and v_j share the same node feature of P_k.

There are many studies on node pattern recognition using GNN (e.g., [27, 31, 34]) of which the process is as follows: given the feature vector x_v of node v in a workflow, $(G, x_v) \rightarrow e_v$ is the embedding of v in which e_v includes its

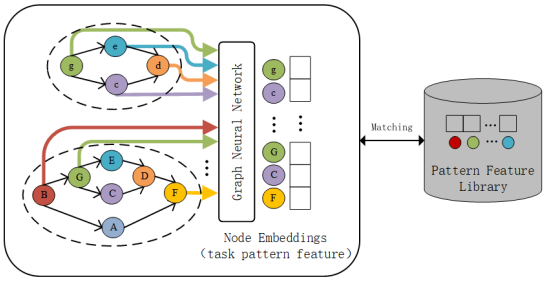


Fig. 3 Node embedding learning for task pattern feature.

neighborhood information, i.e., the aggregated information of its neighbor nodes denoted by $a_v = \sum_{u \in k(v)} f(e_u, c_{v,u})$ in which $k(v)$ is the neighborhood node set of v. e_v is iteratively updated by passing through the GNN network. In each information passing step, e_v is updated by: $e_v = \sum_{u \in k(v)} f(e_u, c_{v,u}) + x_v = a_v + x_v$. In other words, the pattern feature e_v is updated linearly to a_v. No critical path information can be concerned by the existing GNN for microservice workflows.

We develop a function $g(a_v)$ nonlinearly to a_v for updating e_v as follows:

$$e_v = g\Big(\sum_{\mu \in k(v)} f(e_u, c_{v,u})\Big) + x_v = g(a_v) + x_v \tag{9}$$

where both $f(\cdot)$ and $g(\cdot)$ are nonlinear transformations on input vectors. The same nonlinear transformations $f(\cdot)$ and $g(\cdot)$ are repeatedly used to all nodes and all information passing steps. By incorporating the transformation combination of the two nonlinear functions $f(\cdot)$ and $g(\cdot)$ [xx], a wide variety of aggregation functions can be used by the modified GNN model which can obtain critical path information effectively and efficiently. For example, if $f \sim log(\cdot/n)$, $g \sim e^{n\times\cdot}$, and $n \to \infty$, the aggregation $g(a_v)$ obtains the maximum embedding information of its neighbourhood nodes. The two nonlinear transformations can fasten the convergent speed which is verified by experiments in the performance evaluation section. For simplicity, only single-in and single-out patterns are kept in the pattern library in this paper to reduce the computation time of the recognition process.

4.2 Workflow Sequencing

Generally, there are two ways for scheduling multiple workflows: combining all workflows into a single workflow by adding two dummy virtual tasks, scheduling workflows one by one according to the priority policy. Due to the large number of tasks, complex relationships in microservice workflows, the merged workflow is always too large for the first method. Therefore, the proposed framework adopts the second method which schedules microservice workflows according to priorities of the involved workflows.

Two strategies are proposed for workflow sequencing:

- SWS1: The priority of workflow W_i is determined by the urgent degree α_i which is defined as
$$\alpha_i = \frac{D_i - A_i - CP_i}{CP_i} \tag{10}$$
in which CP_i is the length of the critical path of W_i. A smaller α_i means a more urgent W_i. All microservice workflows are sorted in the non-decreasing order of their urgent degrees.
- SWS2: Since patterned tasks are usually more important than non-patterned ones, it is desirable to schedule workflows with more tasks recognized as patterns first. In other words, more patterned tasks imply more information to be reused in the following scheduling process. The pattern coverage rate $\frac{\mu_i^{pattern}}{\mu_i}$ is crucial to determine the scheduling priority of a workflow where $\mu_i^{pattern}$ is the number of nodes with the pattern feature in workflow W_i. However, the workflow with a more urgent degree should be scheduled first. Therefore, the priority can be determined by
$$\beta_i = \max\Big\{\alpha_i, \frac{\mu_i^{pattern}}{\mu_i}\Big\} \tag{11}$$
which considers both the urgent degree and the pattern coverage rate.

An example with four microservice applications is shown in Figure 4. The colored nodes are pattern tasks. The pattern coverage rates of the four applications are 37.5%, 50.0%, 50.0%, 44.4%, respectively. SWS2 means that workflows

with higher pattern coverage have higher priorities when their soft deadlines exceed the execution time of the critical path.

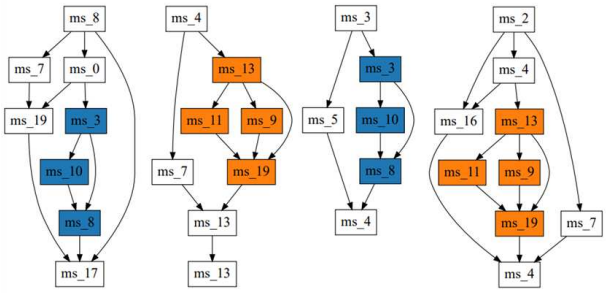


Fig. 4 An example of four workflows with two patterns.

4.3 Task Sorting

Since all tasks of a workflow are constrained by their sub-deadlines, it is crucial to sort these tasks in terms of their sub-deadlines for the scheduling performance. In other words, how to divide the workflow deadline into task sub-deadlines is very important. In this paper, three deadline division strategies are introduced, among which two are traditional and a new one is developed by decomposing the critical path. Patterned nodes are kept during the critical path decomposing process for the first time. The following patterned nodes are inserted to the global sequence directly to reduce the sorting complexity.

The earliest start time, the latest start time, the actual start time, the earliest finish time, the latest finish time, and the actual finish time of $v_{i,j}$ are denoted as $est_{i,j}$, $lst_{i,j}$, $ast_{i,j}$, $eft_{i,j}$, $lft_{i,j}$ and $aft_{i,j}$, respectively. $\varphi^p_{i,j}$ and $\varphi^s_{i,j}$ are the immediate predecessor and immediate successor sets of $v_{i,j}$, i.e., $\varphi^p_{i,j} = \{v_{i,j'}|(v_{i,j'}, v_{i,j}) \in E_i\}$, $\varphi^s_{i,j} = \{v_{i,j'}|(v_{i,j}, v_{i,j'}) \in E_i\}$. $T^e_{i,j}$ is the average execution time of $v_{i,j}$ on all of its candidate resources.

The four static temporal parameters $est_{i,j}$, $eft_{i,j}$, $lst_{i,j}$ and $lft_{i,j}$ can be recursively determined by

$$est_{i,j} = \begin{cases} A_i, & \text{if } j = 1 \\ \max\limits_{v_{i,j'} \in \varphi^p_{i,j}} \{est_{i,j'} + T^e_{i,j'}\}, & \text{if } j \neq 1 \end{cases} \tag{12}$$

$$eft_{i,j} = est_{i,j} + T^e_{i,j} \tag{13}$$

$$lft_{i,j} = \begin{cases} D_i, & \text{if } j = \mu_i \\ \min\limits_{v_{i,j'} \in \varphi^s_{i,j}} \{lft_{i,j'} - T^e_{i,j'}\}, & \text{if } j \neq \mu_i \end{cases} \tag{14}$$

$$lst_{i,j} = eft_{i,j} - T^e_{i,j}. \tag{15}$$

$ast_{i,j}$ is determined only after $v_{i,j}$ is scheduled followed by $aft_{i,j} = ast_{i,j} + T^e_{i,j}$. The interval $[est_{i,j}, lst_{i,j}]$ defines the slack time of task $v_{i,j}$ which implies that $v_{i,j}$ cannot start earlier than $est_{i,j}$ and must start before $lst_{i,j}$ in order to meet the deadline. The time complexity for calculating the four static temporal parameters is $O(|E_i|)$.

Generally, deadline division methods are based on critical paths or partial critical paths. The backward parameter $\ell^B(i,j)$ is the total length from $v_{i,j}$ to the sink node v_{i,μ_i} while the backward parameter $\ell^F(i,j)$ is the total length from $v_{i,j}$ to the source node $v_{i,1}$ of W_i. They can be recursively calculated by:

$$\ell^B(i,j) = \begin{cases} T^e_{i,\mu_i}, & \text{if } j = \mu_i \\ \max\limits_{\forall v_{i,j'} \in \varphi^s_{i,j}} \left\{\ell^B(i,j')\right\} + T^e_{i,j}, & \text{if } j \neq \mu_i \end{cases} \tag{16}$$

$$\ell^F(i,j) = \begin{cases} T^e_{i,1}, & \text{if } j = 1 \\ \max\limits_{\forall v_{i,j'} \in \varphi^p_{i,j}} \left\{\ell^F(i,j')\right\} + T^e_{i,j}. & \text{if } j \neq \mu_i \end{cases} \tag{17}$$

Obviously, the length of the critical path is $\ell^F(i,\mu_i) = \ell^B(i,1)$. The CP-P proposed in [13] utilizes $\ell^B(i,1)$ for deadline division. For the same task $v_{i,j}$, the backward and forward parameters are usually different, i.e., $\ell^B(i,j) \neq \ell^B(i,1) - \ell^F(i,j) + T^e_{i,j}$ is always true for tasks on non-critical paths. Therefore, it is more reasonable to determine the deadline of $v_{i,j}$ by both $\ell^B(i,j)$ and $\ell^F(i,j)$ based on which the average length of $v_{i,j}$ is defined by $\overline{\ell}(i,j) = \frac{\ell^B(i,1) - \ell^B(i,j) + T^e_{i,j} + \ell^F(i,j)}{2}$.

Three deadline division methods are adopted for the considered problem. Besides the traditional PDD (Proportional Deadline Division) and EDD (Equalize Deadline Division), the proposed RSD (Recursive Subgraph Decomposition) decomposes all the tasks of W_i into patterned tasks and non-patterned ones.

- PDD (Proportional Deadline Division): The deadline $d(i,j)$ of each task $v_{i,j}$ is estimated by

$$d_{i,j} = A_i + (D_i - A_i) \times \frac{\overline{\ell}(i,j)}{\ell^B(i,1)} \tag{18}$$

- EDD (Equalize Deadline Division): The deadline of task $v_{i,j}$ is estimated by

$$d(i,j) = A_i + \ell^F(i,j) + (D_i - A_i - \ell^B(i,1)) \times \frac{n(i,j)}{\mu_i} \tag{19}$$

where $n(i,j)$ is the index of $v_{i,j}$ in the task sequence sorted by the non-decreasing order of $\overline{\ell}(i,j)$.

- RSD (Recursive Subgraph Decomposition): Different from the traditional deadline division principles, the proposed RSD decomposes the DAG of W_i into a critical sub-graph W_i^{CP} with only critical tasks and many non-critical sub-graphs $\widetilde{\mathbb{W}}_i = \{\widetilde{W}_i^{(1)}, \widetilde{W}_i^{(2)}, \ldots\}$ with non-critical tasks. W_i is recursively decomposed into a critical sub-graph and non-critical sub-graphs. The deadline of $d(i,j)$ of each task $v_{i,j}$ in the critical sub-graph is determined by Eqn. (18). The available time and deadline of each non-critical sub-graph depends on the latest deadlines of immediate predecessors of its source and sink tasks, respectively.

The task sorting algorithm is shown in Algorithm 2. In the task sorting stage, if the pattern features of the workflow are not considered, the rank of the same pattern needs to be calculated several times. However, considering the pattern information, the order of tasks from the same pattern is order-preserved, i.e., no task sorting is needed from the same pattern for recalculating.

Algorithm 2: PB-TS algorithm

```
for each task v_{i,j} in W_i do
    Calculate the sub-deadline sd_j of v_{i,j}
     by Deadline division strategy
     PDD,EDD or RSD;
Generate Q by sorting tasks in descending
 order by their priorities;
for each v_{i,j} ∈ Q do
    if v_{i,j} is a patterned task then
        Get the sequence Q' of v_{i,j} from
         pattern library;
        Insert Q' to Q;
```

4.4 Resource Selection

In the resource selection stage, the scheduler selects an appropriate container for each task of the task sequence. For patterned tasks, the resource selection process records the container information of historical selection in the pattern library. When the same pattern task is scheduled again, the same resource allocation is preferred. If no container meets the deadline in the preferred allocation, a resource is globally searched. The historical information of non-patterned tasks is not recorded. In an extreme case, no resource can meet the deadline, a container is selected by the EFT. The resource selection process is shown in Algorithm 3.

Moreover, most scheduling algorithms calculate the rank based on the average execution time of tasks on global resources, which always leads to inaccurate estimation on heterogeneous resources as compared to the actual execution time of the final scheduling, especially when the resource space is huge. For the large number of containers in the cloud platform, most low-performance containers could never be selected whereas they are involved in the estimation. Therefore, it is necessary to exclude such the set of resources which can not only reduce the search space in the resource selection stage but also obtain a more accurate average execution performance.

5 Performance Evaluation

5.1 Simulation Setup

Many studies have analyzed the statistical characteristics of microservice applications in data centers [53–55]. Different from the existing benchmark, the size of microservice applications follows a heavy-tailed distribution. According to the DAG graph of the batch application in the Alibaba cluster [56], more than 10% of application call graphs contain more than 40 unique microservices. The largest call graph can even consist of hundreds to thousands of microservices. The average depth of call graph is 4.27, and the derived value of stand is 3.25. The scientific workflow datasets widely used in the previous literature are not suitable for microservice workflow. We generate

Algorithm 3: PB-RS algorithm

```
for each task v_{i,j} ∈ Q do
    if v_{i,j} is a patterned task then
        Get the resource space S from the pattern library in terms of the pattern feature of v_{i,j} ;
        for each container c_k ∈ S do
            Calculate FT(v_{i,j}, c_k) by Eqn. (2);
            if FT(v_{i,j}, c_k) ≤ sd_j then
                π = π ∪ {m_{i,j,k}} ;
            if each container c_k ∈ C then
                Calculate FT(v_{i,j}, c_k) ;
                if FT(v_{i,j}, c_k) ≤ sd_j; then
                    π = π ∪ {m_{i,j,k}};
                    Add c_k to the resource space in pattern library according to the pattern feature of v_{i,j}.
    if v_{i,j} is a non-patterned task then
        for each container c_k ∈ C do
            Calculate FT(v_{i,j}, c_k) ;
            if FT(v_{i,j}, c_k) ≤ sd_j then
                π = π ∪ {m_{i,j,k}}
            c_k = arg min FT(v_{i,j}, c_k);
            π = π ∪ {m_{i,j,k}}.
```

simulation data according to the characteristics of microservice applications, which are randomly generated through the method of literature [56] and ensure that there are sufficient common substructures. For each workflow, the number of tasks is randomly chosen as $\mu_i \in \{30, 50, 100, 150\}$. The multiplying factor is randomly chosen as $\theta \in \{1.3, 1.5, 1.7\}$ which is meaning the deadline of the workflow is θ multiplying the length of the critical path of this workflow application. When calibrating the parameters in the algorithm, a random deadline factor is chosen.

The purpose of multiple experiments is to evaluate the scheduling effectiveness and efficiency under different workflow scale, pattern coverage rate and resource scale. Specifically, the workflow size in one scheduling experiment is set from $n \in \{20, 50, 100, 200\}$. The pattern coverage rate in each workflow is randomly generated from $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The number of heterogeneous containers in the resource space is $\{200, 500, 1000, 2000\}$.

$$D_i = \theta \times CriticalPath(w) \tag{20}$$

The experimental results are analyzed by the multifactor analysis of variance (ANOVA) statistical technique. All the resulting p-values are less than 0.05, indicating that all the studied factors have a significant effect on the RPD response variable at the 95.0% confidence level. Relative Percentage Deviation (RPD) is adopted to evaluate the performance of different algorithms (parameter combinations). The calculation of RPD is defined in equation (21). Let $C_i(A)$ denote the total index from algorithm A on instance i, C_i^* is the smallest total index from all comparison algorithms on instance i. The multifaceted analysis of variance (ANOVA) is used to analyze the results.

$$RPD(A) = \frac{C_i^* - C_i(A)}{C_i^*} \times 100\% \tag{21}$$

5.2 Parameter Calibration

There are three deadline division strategies, and two candidates for workflow sequencing strategies in the PMWSC framework. In order to verify the performance differences of various strategies, the above methods are used to test on random samples.

1) Deadline Division

The Turkey HSD interval with 95% confidence intervals of deadline division rules is shown in Figure 5. It shows that the RPD of EDD is obviously lower than PDD and RSD, which means the fair distribution of slack time to each task in the workflow is more effective in deadline division for the problem.

2) workflow sequencing

The Turkey HSD interval with 95% confidence intervals of workflow sequencing rules is shown in Figure 6. It shows that the RPD of SWS2 is obviously lower than SWS1, which means considering both of urgent degree and pattern coverage rate of microservice workflows is more effective in workflow sequencing.

3) Performance of Pattern Recognizer

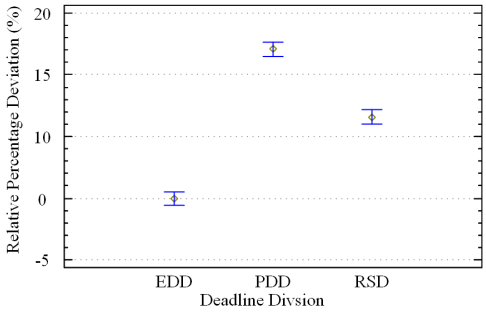


Fig. 5 RPD means plot with 95% Turkey HSD intervals of deadline division rules.

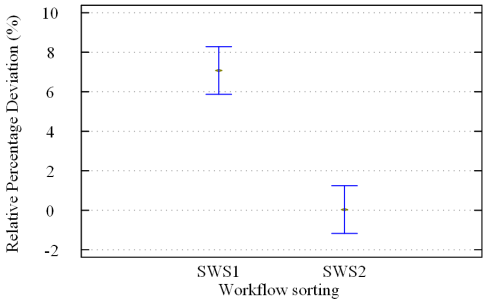
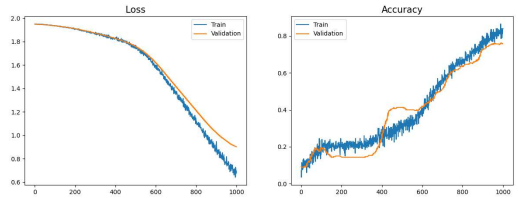


Fig. 6 RPD means plot with 95% Turkey HSD intervals of workflow sequencing rules.

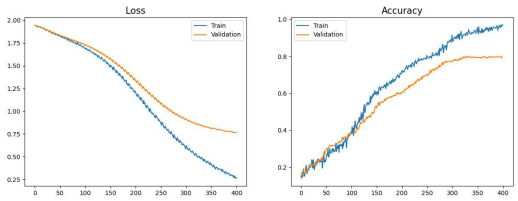
We modify the original GNN model in section 3. The results of loss value and accuracy in the test set and the verification set are shown in Figure 7. It can be seen from the figure that the GNN model achieves an accuracy of 75% in the validation set, but the original model requires 1000 iterations to converge, while the modified model achieves it in around 300 iterations. This may be because the modified model takes into account the characteristics of the dag graph, excludes unnecessary domain nodes in the aggregation function, and identifies single-in or single-out nodes, thereby significantly reducing the dimensionality of node features. Experiments have shown that the improvement makes the GNN model more stable and efficient when dealing with complex dag graph data.

5.3 Algorithm Comparisons

To the best of our knowledge, there is no workflow scheduling problem proposed in previous literature with the same characteristics as the considered problem. In order to test the performance



(a) Original GNN model



(b) Modified GNN model

Fig. 7 Loss and accuracy for train and validation

of the proposed algorithm, the algorithm comparison adopts the deadline division strategy to modify the HEFT and PEFT as HEFT-D and PEFT-D. The reason for choosing these two algorithms for comparison is that HEFT is a classic robust scheduling algorithm in various scenarios, and PEFT is currently the best-improved algorithm for HEFT. Another classic algorithm CPOP determines the priority of a task by the sum of the upward and downward rank values, and arranges critical tasks to the critical path processor to minimize the total execution time of critical tasks. Moreover, PMWSC includes task sorting phase and resource selection phase, which are the same as these three comparison algorithms. In order to test the performance of our framework, which includes the effectiveness and efficiency of scheduling process. The average tardiness is the indicator of the effectiveness, and the indicator of the efficiency is the runtime of the scheduling scheme generated.

5.3.1 Algorithm comparisons for average tardiness

Under the 95% Tukey HSD confidence interval, the comparison result of average tardiness under different workflow size are shown in Figure 8. There is no significant difference between the three algorithms, the RPD value is lower than 0.8%. Our framework is effective compared to the three

algorithms, although its advantages are not obvious. This result proves that our framework will not miss the optimal solution while simplifying the scheduling process.

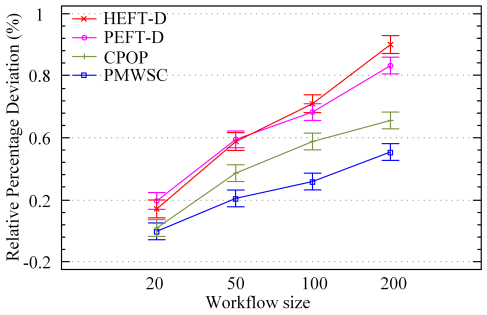


Fig. 8 Comparison of algorithms for average tardiness under different workflow size.

Figure 9 is the comparison result of average tardiness under different pattern coverage rate. It can be seen from the figure that the tardiness of the proposed PMWSC, HEFT-D, PEFT-D ,and CPOP is consistent under different mode coverage. The proposed algorithm has no obvious advantage, which is same effective with two classic algorithms under different pattern coverage rate.

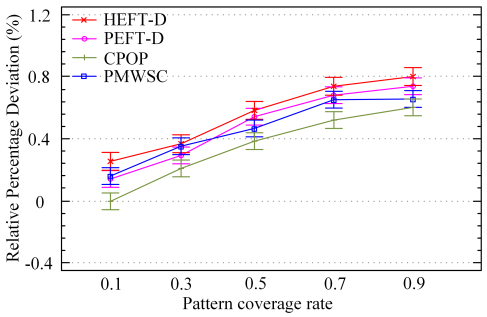


Fig. 9 Comparison of algorithms for average tardiness under different pattern coverage rate.

In order to verify the conclusion that the four algorithms all can find a good solution when the resources are sufficient, the comparison result under different container size is in Figure 10. The result shows that the PMWSC algorithm is not always optimal. When the resources are insufficient, the average tardiness is longer than HEFT-D and PEFT-D. However, when resources are sufficient, the tardiness of PMWSC is equivalent to the three algorithms. Therefore, our framework is more suitable for the environment with sufficient resources.

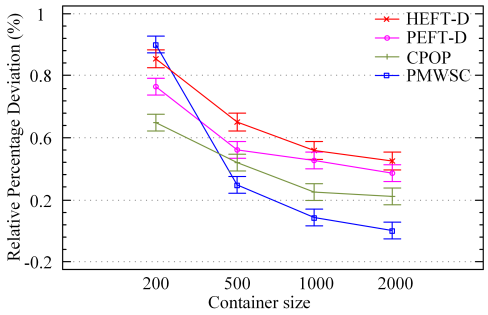


Fig. 10 Comparison of algorithms for average tardiness under different container size.

5.3.2 Algorithm comparisons for runtime

Under the 95% Tukey HSD confidence interval, the comparison result of runtime under different workflow size is shown in Figure 11. Among the these algorithms, PMWSC is the best one. When the workflow number increases, the runtime of these algorithms all increases, but the proposed PMWSC algorithm is always better than others. This is because the proposed algorithm uses similar pattern information to reduce the size of the original problem.

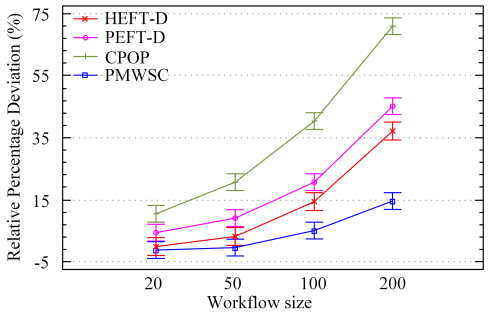


Fig. 11 Comparison of algorithms for runtime under different workflow size.

The average experimental results under different pattern coverage rate is shown in the Figure 12. When the mode coverage rate is low, the proposed algorithm is not as efficient as HEFT-D, PEFT-D and CPOP. When the mode

coverage is about 30%, the execution time of the three classical algorithms is very close, because they do not use the pattern information to make any optimization, and the pattern coverage rate has no obvious impact on the scheduling efficiency. With the increase of mode coverage, the execution time of PMWSC decreases, and the performance of the scheduler is improved.

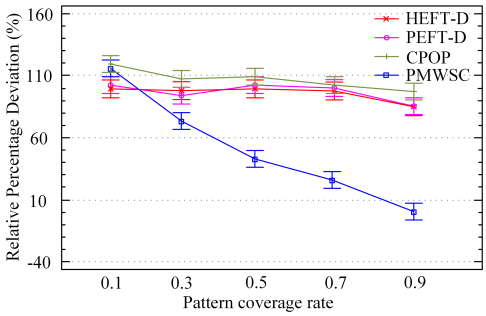


Fig. 12 Comparison of algorithms for runtime under different pattern coverage rate.

Figure 13 is the comparison result of runtime under different container size. Among all the comparison algorithms, PMWSC is the best one. When the container space takes different values, The efficiency of the proposed PMWSC algorithm is better than HEFT-D, PEFT-D and CPOP. It is shown that the proposed algorithm optimizes the container space, thus improving the search efficiency of resources. From the trend in the figure, our framework is suitable for workflow scheduling in massive resource space.

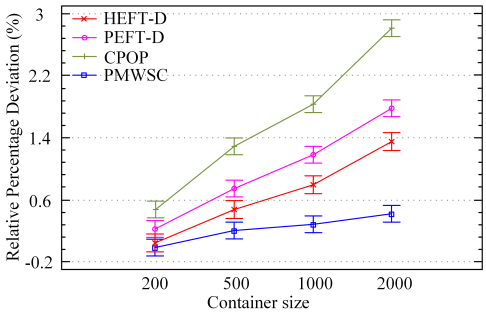


Fig. 13 Comparison of algorithms for runtime under different container size.

6 Conclusion and Future Work

This paper proposes a PMWSC (Pattern-based Microservice Workflow Scheduling to Containers) framework to improve the performance of scheduling microservice workflows with different soft deadlines to heterogeneous containers in cloud computing. Traditional scheduling systems rarely consider the relationship of workflows to improve scheduling performance. Our framework directly extracts the pattern features based on GNN and combines them with the heuristic scheduling process. The experimental results show that the proposed pattern-based workflow scheduling framework leads to better performance, compared to the classical algorithms.

At present, this paper focuses on the optimization of cloud workflow execution efficiency and tardiness, while ignoring the lease cost of cloud resources, and also not considering the automatic scaling of containers. Further research interests include lease cost optimization with personalized budget-constrained, and adopting container auto-scaling technology to improve the flexibility of pattern-based microservice workflow scheduling system.

Acknowledgment

This work is supported by the National Key Research and Development Program of China (No. 2022YFB3305500), the National Natural Science Foundation of China (Nos. 62273089, 62102080), Natural Science Foundation of Jiangsu Province (No. BK20210204), and Collaborative Innovation Center of Wireless Communications Technology.

References

[1] Fowler, M., Lewis, J.: Microservices a definition of this new architectural term. URL: http://martinfowler. com/articles/microservices. html.[Online] (2014)

[2] Roig, E.B.: Building microservices (2017)

[3] Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., Wen, J.: Deadline-constrained cost optimization approaches for workflow scheduling in

clouds. IEEE Transactions on Parallel and Distributed Systems **PP**(12), 1–1 (2017)

[4] Kurhinen, H., et al.: Developing microservice-based distributed workflow engine. (2014)

[5] Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software **33**(3), 42–52 (2016)

[6] Wang, S., Ding, Z., Jiang, C.: Elastic scheduling for microservice applications in clouds. IEEE Transactions on Parallel and Distributed Systems **32**(1), 98–115 (2021)

[7] Adam, O., Lee, Y.C., Zomaya, A.Y.: Stochastic resource provisioning for containerized multi-tier web services in clouds. IEEE Transactions on Parallel and Distributed Systems **28**(7), 2060–2073 (2017)

[8] Venumadhav, A.: A survey of various workflow scheduling algorithms in cloud environment. Ijsrp Org **22**(8), 1483–1496 (2013)

[9] Topcuoglu, H., Hariri, S., Wu, M.-y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE transactions on parallel and distributed systems **13**(3), 260–274 (2002)

[10] Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems **25**(3), 682–694 (2014)

[11] Kanemitsu, H., Hanada, M., Nakazato, H.: Clustering-based task scheduling in a large number of heterogeneous processors. IEEE Press (2016)

[12] Nirmala, S.J., Setlur, A.R., Singh, H.S., Khoriya, S.: An Efficient Fault Tolerant Workflow Scheduling Approach using Replication Heuristics and Checkpointing in the Cloud (2018)

[13] Żotkiewicz, M., Guzek, M., Kliazovich, D., Bouvry, P.: Minimum dependencies energy-efficient scheduling in data centers. IEEE Transactions on Parallel and Distributed Systems **27**(12), 3561–3574 (2016)

[14] Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: e-Science and Grid Computing, 2005. First International Conference On, p. 8 (2005). Ieee

[15] Yuan, Y., Li, X., Wang, Q., Zhu, X.: Deadline division-based heuristic for cost optimization in workflow scheduling. Information Sciences **179**(15), 2562–2575 (2009)

[16] Cai, Z., Li, X., Gupta, J.N.: Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing. In: International Conference on Service-Oriented Computing, pp. 207–221 (2013). Springer

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

[18] Abrishami, S., Naghibzadeh, M., Epema, D.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Generation Computer Systems **29**(1), 158–169 (2013)

[19] Rodriguez, M.A., Buyya, R.: Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. IEEE transactions on cloud computing **2**(2), 222–235 (2014)

[20] Chen, Z.-G., Zhan, Z.-H., Li, H.-H., Du, K.-J., Zhong, J.-H., Foo, Y.W., Li, Y., Zhang, J.: Deadline constrained cloud computing resources scheduling through an ant colony system approach. In: Cloud Computing Research and Innovation (ICCCRI), 2015 International Conference On, pp. 112–119 (2015). IEEE

[21] Chen, Z.-G., Du, K.-J., Zhan, Z.-H., Zhang, J.: Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm. In: Evolutionary Computation (CEC),

2015 IEEE Congress On, pp. 708–714 (2015). IEEE

[22] Li, H., Wang, D., Zhou, M., Fan, Y., Xia, Y.: Multi-swarm co-evolution based hybrid intelligent optimization for bi-objective multi-workflow scheduling in the cloud. IEEE Transactions on Parallel and Distributed Systems **33**(9), 2183–2197 (2022) https://doi.org/10.1109/TPDS.2021.3122428

[23] Melnik, M., Nasonov, D.: Workflow scheduling using neural networks and reinforcement learning. Procedia Computer Science **156**, 29–36 (2019)

[24] Kintsakis, A.M., Psomopoulos, F.E., Mitkas, P.A.: Reinforcement learning based scheduling in a workflow management system. Engineering Applications of Artificial Intelligence **81**(MAY), 94–106 (2019)

[25] Cui, D., Ke, W., Peng, Z., Zuo, J.: Multiple dags workflow scheduling algorithm based on reinforcement learning in cloud computing. In: International Symposium on Intelligence Computation and Applications (2016)

[26] Jiahao, W., Zhiping, P., Delong, C., Qirui, L., Jieguang, H.: A multi-object optimization cloud workflow scheduling algorithm based on reinforcement learning. (2018)

[27] Ma, G., Ahmed, N.K., Willke, T.L., Yu, P.S.: Deep graph similarity learning: a survey. Springer US (3) (2021)

[28] Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs (2017)

[29] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2016)

[30] Shanthamallu, U.S., Thiagarajan, J.J., Song, H., Spanias, A.: Gramme: Semisupervised learning using multilayered graph attention models. IEEE Transactions on Neural Networks and Learning Systems **PP**(99), 1–12 (2019)

[31] Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2017)

[32] Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? (2018)

[33] Sun, Z., Wang, H., Wang, H., Shao, B., Li, J.: Efficient subgraph matching on billion node graphs. VLDB Endowment (2012)

[34] Wang, D., Peng, C., Zhu, W.: Structural deep network embedding. In: Acm Sigkdd International Conference on Knowledge Discovery & Data Mining (2016)

[35] Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph Matching Networks for Learning the Similarity of Graph Structured Objects (2019)

[36] Thönes, J.: Microservices. IEEE Software **32**(1), 116–116 (2015)

[37] Newman, S.: Building Microservices: Designing Fine-grained Systems. " O'Reilly Media, Inc.", ??? (2015)

[38] Sill, A.: The design and architecture of microservices. IEEE Cloud Computing **3**(5), 76–80 (2016)

[39] Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L., Villari, M.: Open issues in scheduling microservices in the cloud. IEEE Cloud Computing **3**(5), 81–88 (2016)

[40] Zheng, C., Thain, D.: Integrating containers into workflows: A case study using makeflow, work queue, and docker. In: Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, pp. 31–38 (2015). ACM

[41] Gerlach, W., Tang, W., Keegan, K., Harrison, T., Wilke, A., Bischof, J., DSouza, M., Devoid, S., Murphy-Olson, D., Desai, N., *et al.*: Skyport-container-based execution environment management for multi-cloud scientific workflows. In: Data-Intensive Computing in the Clouds (DataCloud), 2014 5th International Workshop On, pp. 25–32 (2014).

IEEE

[42] Gerlach, W., Tang, W., Wilke, A., Olson, D., Meyer, F.: Container orchestration for scientific workflows. In: Cloud Engineering (IC2E), 2015 IEEE International Conference On, pp. 377–378 (2015). IEEE

[43] Bhamare, D., Samaka, M., Erbad, A., Jain, R., Gupta, L., Chan, H.A.: Multi-objective scheduling of micro-services for optimal service function chains. In: Communications (ICC), 2017 IEEE International Conference On, pp. 1–6 (2017). IEEE

[44] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., Steinder, M.: Performance evaluation of microservices architectures using containers. In: Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium On, pp. 27–34 (2015). IEEE

[45] Tihfon, G.M., Park, S., Kim, J., Kim, Y.-M.: An efficient multi-task paas cloud infrastructure based on docker and aws ecs for application deployment. Cluster Computing **19**(3), 1585–1597 (2016)

[46] Peinl, R., Holzschuher, F., Pfitzer, F.: Docker cluster management for the cloud-survey results and own solution. Journal of Grid Computing **14**(2), 265–282 (2016)

[47] Li, W., Li, X., Ruiz, R.: Scheduling microservice-based workflows to containers in on-demand cloud resources. In: 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD) (2021)

[48] Hoenisch, P., Weber, I., Schulte, S., Zhu, L., Fekete, A.: Four-fold auto-scaling on a contemporary deployment platform using docker containers. In: International Conference on Service-Oriented Computing, pp. 316–323 (2015). Springer

[49] Wu, H., Hua, X., Li, Z., Ren, S.: Resource and instance hour minimization for deadline constrained dag applications using computer clouds. IEEE Transactions on Parallel and Distributed Systems **27**(3), 885–899 (2016)

[50] Ming, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011 (2011)

[51] Rimal, B.P., Maier, M.: Workflow scheduling in multi-tenant cloud computing environments. IEEE Transactions on Parallel and Distributed Systems **28**(1), 290–304 (2017)

[52] Bao, L., Wu, C., Bu, X., Ren, N., Shen, M.: Performance modeling and workflow scheduling of microservice-based applications in clouds. IEEE Transactions on Parallel and Distributed Systems, 2114–2129 (2019)

[53] Gan, Y., Jackson, B., Hu, K., Pancholi, M., Ritchken, B.: An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: the Twenty-Fourth International Conference (2019)

[54] Sriraman, A., Wenisch, T.F.: μ suite: A benchmark suite for microservices. In: 2018 IEEE International Symposium on Workload Characterization (IISWC) (2018)

[55] Zhou, X., Peng, X., Xie, T., Sun, J., Xu, C., Ji, C., Zhao, W.: Poster: Benchmarking microservice systems for software engineering research. In: IEEE/ACM International Conference on Software Engineering: Companion

[56] Hakim, A.R., Fithriani, I., Novita, M.: Properties of burr distribution and its application to heavy-tailed survival time data. Journal of Physics Conference Series **1725**, 012016 (2021)