Virtual Infrastructure Planning for a Workflow with Multiple Overlapping Deadline Constraints in Cloud

Junchao Wang*, Huan Zhou†, Yang Hu‡, Cees de Laat§ and Zhiming Zhao¶
University of Amsterdam
Email: *{j.wang2, †h.zhou, ‡y.hu, §delaat, ¶z.zhao}@uva.nl

***Abstract*—Cloud providers offer various types of virtual infrastructures (e.g. Virtual Machines, Dockers) to consumers in a pay-as-you go manner. For an application represented by a complex workflow, it is difficult for the consumer to decide what type of infrastructure they need to meet the Quality-of-Service (QoS) requirements and achieve objectives like monetary cost optimisation. We call such problem as the virtual infrastructure planning problem in cloud. Most existing studies focus on only one single deadline-constrained workflow planning. The single deadline constraint usually refers to a global deadline since the start execution of a workflow to its finish. However, such simple model cannot allow a time critical application to specify internal deadlines inside the workflow. To allow such flexibility for consumers, we propose a multi-deadline model and consider the virtual infrastructure planning problem for workflow with multi-deadline constraints. To solve the problem efficiently, we propose a Multiple-Deadline overlapping-based Infrastructure Planning (MDIP) algorithm. In MDIP, we define the criticality of tasks by analysing the deadline coverages. Then we rank the tasks in the workflow by their criticality and assign them with VM type of better performance. To evaluate the effectiveness of our algorithm, we compare it with a meta-heuristic solution (Genetic Algorithm-based) and a hybrid solution combining critical path and genetic algorithm. Simulated experiments show our approach can achieve better results than existing solutions and more efficient than GA-based solution. Keywords: Time critical application, multiple deadlines, workflow scheduling**

I. Introduction

Cloud computing is increasingly popular for hosting applications due to its elasticity and flexibility. These characteristics are mainly reflected in the resource on-demand and pay-as-you-go manner. Deployment time of an application is shrunk and maintenance cost is decreased compared with hosting applications in physical clusters. Due to such advantages brought by the cloud, applications are increasingly migrated to the cloud.

Time critical applications (TCAs) are one specific type of applications that are sensitive to timing constraints, namely deadlines. TCAs usually consist of multiple distributed components which have data dependencies. Without violation of the data dependencies, chained processing of these components to outside data comprises the workflow of a TCA. For instance, a disaster early warning system includes several basic processing components: data pre-processing, simulation and decision making [1]. The data collected from sensors is processed and then fed to the simulation module. Then the simulated data is passed to the decision making component. The whole process of the TCA should finish not exceeding its deadline. Functional correctness is not the only requirement of time critical applications. During the execution of the workflow, the timing requirements of time critical applications should also be satisfied. Failing to meet the deadlines can lead to serious consequences. For instance, if a disaster early warning system cannot report a disaster in time, massive physical infrastructures can be destroyed. When the response time of an interactive video broadcasting system increases, the user experience is strongly degraded. The system can therefore miss a lot of consumers.

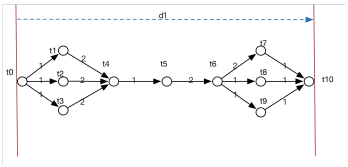
When migrating a time critical application to the cloud, users should decide the right type of Virtual Machines (VM) for each component and schedule the tasks in accordance with their data dependencies. The planned infrastructures should guarantee all the deadline requirements of the application and achieve objectives like cost minimisation. We call such problem as the "Virtual Infrastructure Planning Problem (VIPP)".

There are existing research studying the virtual infrastructure planning problem for workflows with single deadline requirement [2] [3] [4]. The deadline refers to a global deadline from the start to the end of the workflow's execution. However, such constraint is not adequate for TCAs with multiple deadline requirements and constrains the user specifying some internal deadlines. For example, the Internet-of-Things (IoT) applications are quite complex and called system of systems. Each component of the application is also an individual system. The whole application has its deadline requirement but some components also have their own deadline requirements. In our previous work, we studied the problem of VIPP for TCAs with multi-deadline requirements [1]. The multiple deadline model only allows users to specify deadlines since the start of a workflow. In this paper we relax such constraints that only allows users to specify deadlines between any two components inside the workflow.

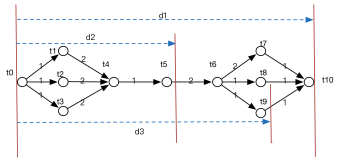
Three examples of different deadline models listed in Fig. 1 are shown below. Fig. 1a shows an example of a workflow with single deadline constraint. Fig. 1b shows a multi-deadline constrained model. The model requires all the deadlines starting from the entry task of a workflow. Fig. 1c shows a more flexible multi-deadline model that allows user to specify a deadline between any two tasks inside a workflow.

TABLE I: Caption for the table.

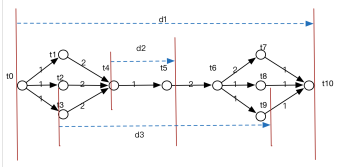
Problem	Solution		constraints
Workflow planning with singe deadline (shown in 1a)	IC-PCP Genetic Algorithm CPI [4]	[2] [3]	Single deadline solutions cannot be directly adapted to solve the multiple deadline problem
Workflow planning with constrained multiple deadlines (shown in 1b)	MEPA [1]		Scalability problem; Multiple deadline flexibility issue
Workflow planning with multiple overlapping deadlines (shown in 1c)	MDIP		Not global optimal



(a) VIPP for workflow with single deadline requirement



(b) VIPP for workflow with constrained multiple deadline requirements



(c) VIPP for workflow with multiple deadline requirements

Fig. 1: Example for VIPP with deadline requirements

II. Related Works

Yu et al. [5] propose a sub-deadline method to minimise the execution cost of a workflow and satisfy a global deadline. They distribute the global deadline to each task in the workflow by analysing the structure of the workflow. The IaaS Cloud Partial Critical Paths (IC-PCP) algorithm [2] is a critical path-based algorithm that first assigns all the tasks in the workflow with the fastest VM type. Then it calculates the partial critical paths in the workflow and assigns VM types without violating the tasks' LFT (Latest Finish Time). The Critical Path-based Iterative (CPI) [6] and complete Critical Paths (CPIS) [4] algorithms are other algorithms for solving the cloud infrastructure planning problem within the bounds of a single deadline. Instead of computing the partial critical path, they identify the complete critical path in a workflow and update resource assignment without violation of the workflow's deadline. Rodriguez et al. [3] apply particle swarm optimisation (PSO) by encoding the task-resource mapping as the particle's position. In this paper, we apply a similar meta-heuristic solution, Genetic Algorithm-based solution as the baseline to compare the results given in this paper. Convolbo and Chou [7] apply a heuristic approach through analysing the parallism of a workflow. Wu et al. [8] propose a heuristic algorithm by applying minimal slack time and minimal distance to guarantee the global deadline of the workflow and then a VM instance hour minimisation algorithm is applied to further reduce the cost. However, all these existing research focus on only one global deadline. The multi-deadline VIPP problem which is quite common in industrial use cases is not addressed.

III. Problem Formulation

TABLE II: Caption for the table.

Notation and Abbreviations	Meaning
TCA	Time Critical Application
V	tasks in a workflow
E	communication between tasks
Q	set of multiple deadline requirements
B_G	cost of executing the workflow
Z_G	makespan of the workflow
$pred(v_i)$	predecessors of v_i
$succ(v_i)$	successors of v_i
MMKP	Multiple-Choice Multi-Dimensional Knapsack Problem
MDIP	Multiple overlapping Deadlines Infrastructure Planning
VIPP	Virtual Infrastructure Planning Problem

In this paper we use $G = \langle V, E \rangle$ to represent the workflow of an TCA. $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes that corresponds to tasks in workflow G. E represents the communication between tasks. For each task $v \in V$, we define the parents of v as $pred(v) = \{v' \mid v' \in V \wedge (v', v) \in E\}$. $\forall v_i \in V$, v_i cannot start executing until all its predecessors $pred(v_i)$ finish. Correspondingly, we define the children of v as $succ(v) = \{v' \mid v' \in V \wedge (v, v') \in E\}$. We assume that the tasks in workflow G can be executed on different types of VM services provided by the cloud provider and cannot be split into two or more sub-tasks. The communication links between the tasks in V are represented by E such that $\forall e \in E$, e is a tuple (v, v') where $v \in V$ denotes the source of the communication and $v' \in V$ denotes the destination of the communication. $w(v, v')$ represent the communication cost between v and v' in the workflow. The communication cost denotes the data transfer time between one task to another. We also assume, for convenience, that every workflow has a single initial task v_{entry} such that $pred(v_{entry}) = \emptyset$ but $pred(v) \neq \emptyset$ for all other tasks $v \in V$. Similarly, we assume

that every workflow has a single terminal task v_{exit} such that $succ(v_{exit}) = \emptyset$ but $succ(v) \neq \emptyset$ for all other tasks $v \in V$.

A cloud provider often offers different types of VM service at different prices for customers to choose from; e.g., M (general purpose), I (I/O optimised), C (computing optimised) and R (memory optimised) VMs as offered by Amazon EC2 [9]. In this paper we denote such VM services as basic service types. Each task in the workflow can be deployed on an instance of one VM service type. When we refer to a *VM type*, we refer to a VM service type offered by the cloud provider. We refer to a concrete VM to which a single task is assigned as a *VM instance*. Assume the cloud provider provides m types of VM service $s_1, \ldots, s_m$, and that the price per time unit of each service s_i is c_i. Deployment of tasks on different VM services will result in different performance, which can be represented by a performance matrix T. Each element $t_{ij} = T[s_i, v_j]$ is the execution cost of task v_j on service s_i, being the length of time between the arrival of a request and the generation of the corresponding response.

We use Q to represent the set of deadline requirements of the time critical application workflow, where $q = \langle v, v', d$ where $q \in Q$ and $v, v' \in V$. q denotes that task v_j should finish before time d since the start v to the finish of v'. Compared with our previous work [1], this QoS definition allows more flexibility. Our previous work only allows users specify QoS from the entry node to another internal node in the workflow. In this paper we break such limits by allowing users to specify deadline between any two nodes in the workflow. In this model, a single global application deadline can be seen as a special case: if a workflow has only a global deadline d_1, then the QoS requirement of the workflow is $Q = \{\langle v_{entry}, v_{exit}, d_1\}$ where v_{entry} is the entry task and v_{exit} is the finish task in the workflow. The multi-deadline model in our previous work can also be seen as a special case where the deadline denotes the maximum execution time between the entry task to internal tasks inside the workflow. In this paper we use B_G (See Algorithm 1) to represent the cost of the workflow G. Formally, $B_G = \sum_{i=1}^{n} \sum_{j=1}^{\alpha} (c_{ij} \times x_{ij})$.

$$minimize B_G \tag{1}$$

subject to

$$\forall q \in Q, \Phi(q) \prec p \tag{2}$$

$$\sum_{j=1}^{m} x_{ji} = 1 \tag{3}$$

$$x_{ji} \in 0, 1 \tag{4}$$

We use $EST(v_i)$ and $EFT(v_j)$ to denote the *Earliest Start Time* (EST) of v_i and *Earliest Finish Time* (EFT) of v_j. We use $A(v_i)$ to represent that the task v_i is assigned with the $A(v_i) = s_j$ type of VM. The Earliest Start Time (EST) of task v_i represents that during the processing of the workflow, v_i can start processing an event at its EST. When all the tasks in the workflow have been assigned, the Earliest Start Time of task v_i is defined as follows [2]:

$$EST(v_{entry}) = 0 \tag{5}$$

$$EST(v_i) = \max_{v_p \in pred(v_i)} \{EST(v_p) + T[A(v_p), v_p] + w(v_p, v_i)\} \tag{6}$$

Accordingly, the Earliest Finish Time (EFT) of v_i is defined as:

$$EFT(v_i) = EST(v_i) + T[A(v_i), v_i] \tag{7}$$

A. Problem Analysis

Theorem 1. *A VIPP for a sequential workflow with multiple deadline constraints is equal to the Multiple-Choice Multi-Dimensional Knapsack Problem (MMKP).*

A sequential workflow refers to a workflow that each node $v_i \in V$
v_{entry}, v_{exit} has only one predecessor and one successor. Formally, $|pred(v_i)| = 1$ and $|succ(v_i)| = 1$.

Proof. The Multiple-Choice Multi-Dimensional Knapsack Problem (MMKP) is a variant of the famous 0-1 knapsack problem [10]. Unlike the simple 0-1 knapsack problem, MMKP considers n classes where each class has J_i items. Users can choose an item for each class. The VIPP is actually an infrastructure-to-workflow mapping problem. Each task in the workflow is mapped to one type of VM. Thus, we can consider each task in the workflow equally as the n classes in MMKP. The number of VM types can be considered as choices for each class.

In MMKP, there are several resource requirements. While selecting items in each class, the total profit value of the selection should be maximised while subjecting to the resource constraints. For a sequential workflow, the execution time between any two tasks can be simply calculated by summing up the processing time of tasks in-between. Thus, the resource requirements in MMKP can be theoretically equal to the multiple deadline requirements of the VIPP. Therefore, the MMKP is theoretically equal to the VIPP.

□

Since MMKP is NP-hard, the VIPP is NP-hard. In 1, we prove a VIPP for a sequential workflow is equal to the MMKP. There is only one path from the entry task to the exit task of the sequential workflow. However, in time critical applications, workflows are not only simple pipelines between tasks but have complex data dependencies. The tasks also exhibit diverse performance characteristics on different types of VMs. Therefore, the sequential workflow planning problem can be seen as a special case in VIPP.

IV. Multiple overlapping Deadlines Infrastructure Planning (MDIP)

There are approximate solutions to the MMKP under polynomial time. However, since the VIPP in this paper considers not only the simple sequential workflow, but also the complex workflows. Existing solutions to MMKP is not appropriate for solving VIPP. Therefore, we propose a Multiple overlapping Deadlines Infrastructure Planning (MDIP) algorithm to solve the VIPP by analysing the workflow and its multiple deadline requirements.

Pseudo code of the MDIP is shown in Algorithm 1. Generally the MDIP applies a bi-directional strategy. To find an optimal solution for VIPP, it is a searching process. According to 1, theVIPP is NP-hard and quite complex. The searching process can be time consuming when the scale of workflow increases. The initial of the searching becomes very crucial to the quality and searching time of the final solution. Thus, we apply two heuristics for the initialisation of the searching process: $worstStartSearch$ and $bestStartSearch$. The $worstStartSearch$ initialises all the tasks in the workflow with the slowest processing VM type while the $bestStartSearch$ initialises all the tasks with the fastest processing VM type. These are two extreme solutions to the VIPP. In public clouds such as Amazon EC2, Microsoft Azura, the VM type with better performance is more expensive. Assigning all the tasks in the workflow with the fastest VM type may guarantee the multiple deadline requirements, but it can lead to higher monetary cost. As illustrated in our problem formulation, we try to minimise the cost while guaranteeing the multiple deadline requirements. $worstStartSearch$ is more appropriate for "loose" deadline because the initialisation is closer to the optimal solution; the $bestStartSearch$ is more appropriate for "tense" deadline. The MDIP combines the two initialisations and compare their results to choose a better one.

Algorithm 1: MDIP ALGORITHM

Input: G
Output: Planned Infrastructures
cost1, infs1 = worstStartSearch(G)
cost2, infs2 = bestStartSearch(G)
if *cost1* $>$ *cost2* **then**
    return infs1
else
    return infs2

The only difference between $worstStartSearch$ and $bestStartSearch$ is the initialisation. Thus, we only illustrate the $worstStartSearch$ in the following part of this paper.

A. Definitions

Before we introduce the $worstStartSearch$ algorithm, we need to introduce three definitions.

Definition 1. *We use VM quality referring to the processing time of a task on a certain type of VM. For task v, if its processing time on VM type s is faster than s', we call s is better than s'. Correspondingly, if v's processing time on s is slower than s', we call s is worse than s'.*

In the above definition, it should be noticed that the processing time is related to the VM type and task. Usually in the cloud, if a task v performs better on a VM type s than another VM type s', other tasks can also be faster on VM type s than s'. But there can be situations that a better VM type will not always perform better than another. For instance, existing cloud providers offer VM types intended for memory or network optimisation. Memory intensive tasks can perform better on memory-optimised VMs but may not perform as well on network-optimised VMs. In this paper we use Definition 1 to allow more flexibility.

Definition 2. *For deadline $q = (v_i, v_j, d)$, we define its range to be $range(q) = U\{v|v \in simplePath(v_i, v_j)\}$.*

$simplePath(v_i, v_j)$ refers to the simple paths without loops from v_i to v_j. A deadline's range denotes the tasks it covers and takes effect when planning VM types for these tasks.

Definition 3. *The task criticality for task v is defined as $|\{q|q \in Qandv \in range(q)\}|$.*

Task criticality refers to the number of deadlines whose range it is in. When a task is in the range of more deadlines, its VM customisation can have a larger effect on whether more deadlines can be met. Thus, the task with higher task criticality should be considered first.

As shown in Fig. 1c, deadline range of d_1 is $\{t_0, t_1, \ldots, t_10\}$. Deadline range of d_2 is $\{t_4, t_5\}$. Deadline range of d_3 is $\{t_3, t_4, t_5, t_6, t_9\}$.

As we can see from the figure, t_4 and t_5 are in the range of the three deadlines. So the task criticality for these deadlines is the highest.

B. Worst-Start-Search

Pseudo code of the $worstStartSearch$ is shown in Algorithm 2. Initially we calculate the deadline range of each deadline. Then we calculate the criticality of each task based on the deadlines' range. By assigning the worst type of VM to each task, we calculate the makespan of the workflow. The tasks are then sorted according to their criticality. For each task, we assign the task to a better VM if the deadline cannot be met. This process will proceed until all the deadlines can be met or all the tasks are visited.

In the upgrading part of the VM type for a task v, we apply a binary searching strategy to find the VM type that can meet a deadline and enhance the performance of our algorithm. The makespan calculation algorithm is designed to calculate the Earliest Start Time (EST) and Earliest Finish Time (EFT) of tasks in the workflow. We apply a Depth-first heuristic to calculate the EST and EFT of tasks in the workflow from the entry task. Specifically, pseudo code of the makespan

Algorithm 2: WORST-START-SEARCH ALGORITHM

Input: G
Output: Planned infrastructures

```
for q in Q do
    Calculate range(q) according to Definition 2.
for v in V do
    Assign each v with the best VM type according to Definition 1. for q in Q do
        if v ∈ range(q) then
            v's task criticality (Definition 3) increase by 1.
calMakespan(v_entry)
Sort the tasks in V according to their criticality.
for v ∈ sorted(V) do
    Rank the VM types for v according to Definition 1.
    for q(v_i, v_j, d) in Q do
        while EFT(v_j) − EST(v_i) ¿ d or VM type is not the best for v do
            Upgrade v to a better VM type through binary searching.
            Update the EFT, EST affected by v.
Return VM assignments to v.
```

calculation algorithm is shown below. The algorithm applies a recursive function by visiting all the successors of a given node v.

Algorithm 3: LFTCALCULATION

Input: G, v, T, W, Q
Output: Updated of all v's successors

```
if succ(v) = ∅ then
    return
else
    for v_p ∈ succ(v) do
        EST = EFT(v) + W[v, v_p]
        if EST > EST(v_p) then
            EST(v_p) = EST
            EFT(v_p) = EST + T[A(v_p), v_p]
            LFTCalculation(G, v_p, T, W, Q)
```

C. Time Complexity Analysis

Time complexity of CPI is $\mathcal{O}(N^2DM)$ [6], where N refers to the number of tasks in a workflow. D is the global deadline of the workflow. M is the number of types of cloud VM offerings. Time complexity of IC-PCP is $\mathcal{O}(log_2NMN)$ [2]. Time complexity of GA is $\mathcal{O}(kPop * NM)$, where k refers to the iterations of GA and Pop refers to the size of the population in GA. Time complexity of MEPA is $\mathcal{O}(kPop*log_2NM)$, where k refers to the iterations of GA used in MEPA and Pop refers to the size of the population in MEPA. In MEPA, since we assign a partial critical path instead of the whole workflow, the k and Pop are relatively much smaller than that of in GA. However, when the workflow becomes a sequential workflow, the time complexity for MEPA and GA become the same. According to Algorithm 2 and Algorithm 1, time complexity of MDIP is $\mathcal{O}(NM * |Q|)$.

However, CPI, IC-PCP are mainly designed to tackle the single deadline-constrained VIPP. MEPA is able to deal with a special kind of VIPP with multiple-deadline requirements. GA can help to tackle the multiple deadline constrained VIPP. But since GA is notorious in its time complexity, the algorithm converges rather slow when dealing with large scale workflows.

V. EXPERIMENTS

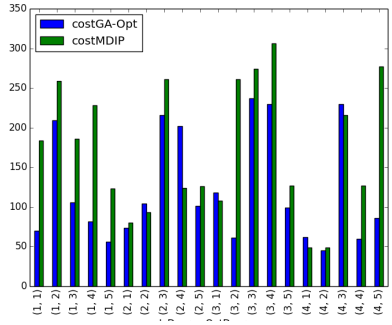
To our knowledge, there is no existing solution targeting at solving the multi-deadline VIPP problem in this paper. Thus, we mainly evaluate the deadline missing rates for single deadline VIPP solutions and cost for multiple deadline solutions.

A. Workload generation

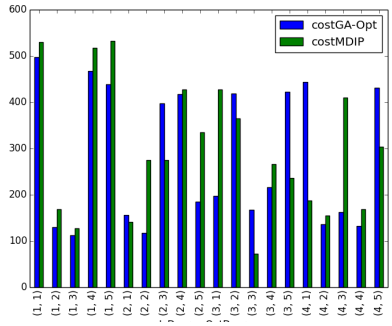
In this paper we use randomly generated workflows to evaluate the performance of our proposed solution. The random generation methods include: fan-in, fan-out and the density-based method. The random methods will always be added with an entry and exit node. For each workflow, we use the "Range Based ETC Matrix Generation" method proposed in [11] to generate the average task execution time. Each value in the matrix is also considered as a task's execution time. We set the number of types of VMs randomly from [3,5,8]. As observed from the instance price of Amazon EC2 [12], we found that the price almost doubles from the previous instance type. So we set the price for instance types as: 1, 2, 4, exponentially. The total number of deadlines is set in positive portion to the total number of nodes in the graph.

B. Experimental results on small scale workflows

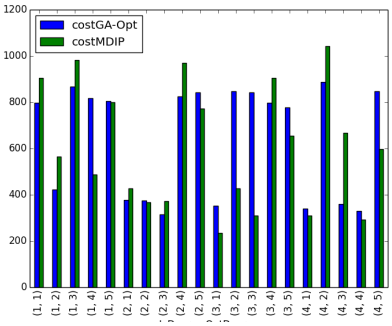
The figures below show cost comparison given by the GA-Opt algorithm and MDIP. The x-axis represents the workflow's maximum in-degree and out-degree. GA-Opt algorithm is an algorithm that applies a heuristic in the initialisation phase by generating the solutions with the same type of VM for all tasks in the workflow. From the results we can see that MDIP proposed in this paper can achieve solutions with slightly higher cost than GA-Opt in some cases. This is mainly due to the deadlines' distribution in the workflow. Although some tasks in a workflow has a higher task criticality defined in Definition 3, they can be small tasks that have little effect on the makespan of the workflow. So assigning these tasks with better VM may not guarantee the multiple deadlines.



(a) Workflow with 16 nodes

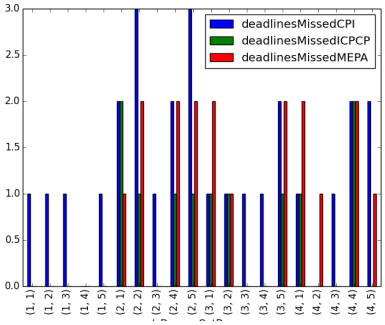


(b) Workflow with 32 nodes

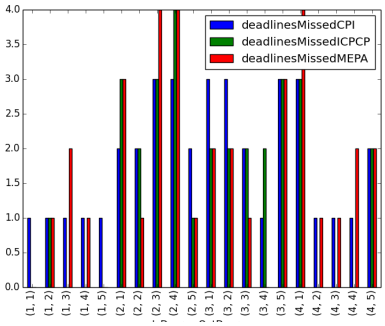


(c) Workflow with 64 nodes

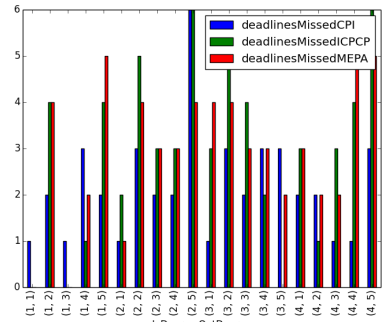
Fig. 2: Cost comparison with 5 VM Types



(a) Workflow with 16 nodes



(b) Workflow with 32 nodes



(c) Workflow with 64 nodes

Fig. 3: Deadlines missed comparison with 5 VM Types

Fig. 3 below show the number of deadline missed with solution given by CPI, ICPCP and MEPA. The MDIP solution proposed in this paper can guarantee all the deadlines, but CPI, ICPCP and MEPA can fail in some internal deadlines. In cases like workflow of 16 nodes with in-degree of 1 and out-degree of 1, the ICPCP and MEPA are able to produce results without violation of any deadlines. But CPI's result fails in one deadline. We can also observe that the deadline can be guaranteed When the scale of the workflow is small, CPI performs worse than ICPCP and MEPA. For workflows with 16 nodes, the CPI only performs better than ICPCP and MEPA in three cases (in-degree of 4 and out-degree of 2, in-degree of 3 and out-degree of 1, in-degree of 4 and out-degree of 1).However, when the scale of the workflow increases, ICPCP performs worse than CPI and MEPA in most cases.

VI. Conclusion and Future Works

In this paper we propose a heuristic solution for planning a virtual infrastructure for workflow with multiple deadline requirements. We define deadline range and task criticality in our algorithm and use them as priority to assign VM types. We apply a bi-directional heuristic to search for the optimal mappings from workflow to VM types. We prove our algorithm is more scalable than Genetic-Algorithm based solution through theoretical analysis. Experimental results show that our proposed MDIP algorithm can achieve almost the results as good as GA-based solution.

For the VIPP problem, different solutions have different characteristics. In our future work we will build a systematic method to advise the most appropriate algorithm to the user by analysing a workflow and its QoS requirements.

ACKNOWLEDGEMENTS

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) and 676247 (VRE4EIC project).

REFERENCES

[1] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, and Z. Zhao, "Planning virtual infrastructures for time critical applications with multiple deadline constraints," *Future Generation Computer Systems*, 2017.

[2] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[3] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 222–235, 2014.

[4] Z. Cai, X. Li, and J. N. Gupta, "Heuristics for provisioning services to workflows in xaas clouds,"

[5] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *e-Science and Grid Computing, 2005. First International Conference on*, pp. 171–200, IEEE, 2005.

[6] Z. Cai, X. Li, and J. N. Gupta, "Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing," in *Service-Oriented Computing*, pp. 207–221, Springer, 2013.

[7] M. W. Convolbo and J. Chou, "Cost-aware dag scheduling algorithms for minimizing execution cost on cloud resources," *The Journal of Supercomputing*, vol. 72, no. 3, pp. 985–1012, 2016.

[8] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and instance hour minimization for deadline constrained dag applications using computer clouds,"

[9] "Amazon ec2 product details." Accessed: 2016-3-29.

[10] A. Sbihi, M. Hifi, and M. Michrafy, "Algorithms for the multiple-choice multidimensional knapsack problem.," *Les Cahiers de la MSE: sÃ© rie bleue*, vol. 31, 2003.

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

[12] "Ec2 instance pricing." Accessed: 2015-12-8.