Planning Virtual Infrastructures for Time Critical Applications with Multiple Deadline Constraints

Junchao Wang[a,b], Arie Taal[a], Paul Martin[a], Yang Hu[a], Huan Zhou[a], Jianmin Pang[b], 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*

Abstract

Executing time critical applications within cloud environments while satisfying execution deadlines and response time requirements is challenging due to the difficulty of securing guaranteed performance from the underlying virtual infrastructure. Cost-effective solutions for hosting such applications in the Cloud require careful selection of cloud resources and efficient scheduling of individual tasks. Existing solutions for provisioning infrastructures for time constrained applications are typically based on a single global deadline. Many time critical applications however have multiple internal time constraints when responding to new input. In this paper we propose a cloud infrastructure planning algorithm that accounts for multiple overlapping internal deadlines on sets of tasks within an application workflow. In order to better compare with existing work, we adapted the IC-PCP algorithm and then compared it with our own algorithm using a large set of workflows generated at different scales with different execution profiles and deadlines. Our results show that the proposed algorithm can satisfy all overlapping deadline constraints where possible given the resources available, and do so with consistently lower host cost in comparison with IC-PCP.

Keywords: Cloud computing, workflow planning, time critical, multiple deadline, partial critical path, QoS.

1. Introduction

Time critical applications are often complex systems with distributed components, dynamic behaviours and high quality of service or experience (QoS/QoE) requirements. An application such as a disaster early warning system is not only required to collect and analyse the influx of sensor data in real time, but it also has to be able to make rapid decisions based on sudden events and adapt gracefully to increased activity. Meanwhile, a live event broadcasting system may represent a more stable environment, but it also needs to process and switch between video feeds while maintaining a constant low level of latency over a long period of time. Such applications are often distributed and modelled as distributed workflows with multiple internal time constraints where, aside from the global deadline for the entire workflow, certain sub-tasks have local deadlines for responding to input.

Strict requirements for runtime infrastructure often make the development and operation of time critical systems difficult and expensive. Cloud computing delivers all the traditional software and physical infrastructure entities as services, and provides a common interface for cloud users in a pay-as-you-go manner. The flexibility and elasticity of cloud computing have enabled it to emerge as a major trend in computer science leading to its wide adoption in both academic and industrial areas. These features also motivate the migration of time critical applications to the cloud; however, there are still great challenges facing developers who wish to build cloud infrastructures that guarantee systematically high performance for applications [1]. Most time critical applications are a composition of different tasks with complex data dependencies. It has already been shown that deployment of the same task on different kinds of resource offered by a cloud provider can lead to different results [2–4]; fundamentally, better virtual infrastructure

*Corresponding author
Email addresses: j.wang2@uva.nl (Junchao Wang), a.taal@uva.nl (Arie Taal), p.w.martin@uva.nl (Paul Martin), y.hu@uva.nl (Yang Hu), h.zhou@uva.nl (Huan Zhou), pang.jianm@gmail.com (Jianmin Pang), delaat@uva.nl (Cees de Laat), z.zhao@uva.nl (Zhiming Zhao)

can lead to better performance, but at greater cost. An infrastructure planner for the Cloud therefore should plan an optimal infrastructure that not only meets the QoS requirements of the application, but that also achieves additional objectives such as minimising monetary cost or power consumption. For this reason, cloud infrastructure planning for time critical applications is often more challenging than scheduling application workflows onto fixed infrastructure.

Our concern here is first and foremost with the selection of virtual infrastructure that is capable of running a distributed application workflow such that all internal response time deadlines will be met. We focus on *persistent* workflows, where the constituent tasks are implemented as services that will continue to run for the entire application lifespan, and thus must continue to respond to new input under the same deadline constraints throughout that lifespan. We assume that the infrastructure resources selected exhibit a stable level of performance, with a stable price for purchasing those resources that is maintained for the entire duration of an application's execution.

In order to cope with the challenges of time critical applications in the Cloud, we propose a **Multi-dEadline workflow Planning Algorithm (MEPA)** to plan the most cost-effective virtual infrastructure for an application workflow with multiple internal deadlines, aimed at guaranteeing all response time deadlines and minimising the required operating budget. To measure the performance of MEPA, we adapt the popular IC-PCP algorithm [3] in order to better fit with the specific characteristics of our problem and use the adapted algorithm as our measurement baseline.

The rest of the paper is organised as follows: section 2 reviews related work, section 3 describes the workflow planning problem, section 4 presents the details of MEPA, section 5 describes the workload used for experiments, section 6 presents our experimental results, section 7 concludes the paper and section 8 presents possibilities for future work.

2. Related Work

For the problem of planning a virtual infrastructure for distributed applications, we focus on the abstract *application workflow*, the decomposition of an application into individual discrete tasks and communication dependencies, and the assignment of individual tasks in the workflow to virtual resources (generally virtual machines or VMs) provided by a cloud environment. In formulating our problem and constructing a solution, we can draw upon prior work in the domains of e-science (which has long been concerned with the deployment of scientific workflows on grid or cloud computing environments), and the development of algorithms for producing optimal resource assignments on virtual infrastructures.

Deelman et al. [5] provide a survey of the different kinds of workflow found in the e-science domain. Based on their analysis, we can classify workflows deployed on virtual infrastructure into two basic categories: scientific workflows and service workflows. Scientific workflows are workflows in which each task is executed once and the virtual resource on which the task is deployed is released upon completion of the task and all following communication between the task and its successors. Service workflows are those with tasks that can be regarded as persistent services, where the tasks persist until the whole application is completed, and have to continue to respond to new inputs for the entire duration of the application. Workflows in both categories may exhibit multiple deadlines, but our concern in this paper is with the latter kind of workflow, which are often used for time critical applications in environmental monitoring. UrbanFlood [6] is an example of an early warning system that tries to solve the problem of flood control, while Kosukhin [7] presents an architecture for performing extreme metocean event forecasting on cloud platforms. In the case of the UrbanFlood system, the workflow has multiple stages with separated modules for sensor monitoring, AI anomaly detection, reliability analysis, breach simulation, virtual dikes, and decision support. Such a system can have multiple internal deadlines in order to ensure timely responses, especially if individual modules must report to other external systems; the quality of service is not addressed in [6] when planning the infrastructure for the application however.

Allocating and scheduling cloud resources for application workflows has become increasingly important for both the cloud provider and application developer, and so there are now many scheduling algorithms available to determine the amount and type of virtual machines needed to execute such workflows at minimal cost. To the best of our knowledge however, all this work addresses the problem of planning infrastructures for workflows that have a single global single deadline, rather than multiple internal deadlines which is our main concern.

Yu et al. [8] propose a method to minimise the execution cost of a workflow to satisfy a global deadline. Their method first clusters the sequential tasks that have only one parent and child together and assigns each task with a sub-

deadline based on its minimum processing time and the sub-deadlines of its predecessors. Each task is then assigned to the cheapest VM that can meet the deadline. However, the communication cost between tasks is not considered.

The IaaS Cloud Partial Critical Paths (IC-PCP) algorithm [3] calculates partial critical paths through the application workflow in order to schedule tasks in the cloud in order to solve the same problem. One assumption IC-PCP makes is that when two tasks are executed on the same VM, the data communication cost between those tasks is zero. This assumption was widely used and discussed for scheduling in heterogenous computing systems with unbounded numbers of processors [9, 10]. In cloud systems, this assumption is quite reasonable because the bandwidth of two tasks communicating in the same VM is typically far higher than the bandwidth between two different VMs. This assumption can be adjusted however by adding a ratio reduction on the communication time instead of setting it to zero [4].

IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2) [3] combines IC-PCP with the approach taken by [8]; after finding a partial critical path, each task in the path is assigned a sub-deadline with the execution time in proportion to the whole partial critical path length. The tasks in the workflow are then assigned with the cheapest VMs.

Meta-heuristic approaches for minimising the execution cost of workflows have also been proposed. Rodriguez et al. [4] apply particle swarm optimisation (PSO), encoding the task-resource mapping as the particle's position. They designed a schedule generation algorithm to decode the encoded particle's position into a schedule by calculating the starting time based on data and resource dependencies (for example the sharing of a single VM by multiple tasks). It is difficult to determine whether the two tasks are provisioned in the same VM only based on the resource type information.

Instead of explicitly considering the deadline as a constraint, Convolbo and Chou [11] only minimise the execution cost of the workflow and propose a heuristic approach which exploits the parallel properties of the workflow. They then adopt a layering approach to determine the schedule of tasks and VM type in each layer. However, communication cost is not discussed in [11], either.

Heterogeneous Earliest Finish Time (HEFT) has been proved to perform better than other heuristics in robustness and schedule length [12]. In [13], Durillo and Prodan propose Multi-Objective HEFT to optimise the trade-off between monetary cost and makespan of the workflow by extending HEFT; the communication cost is only used in the task ranking phase and not addressed in the infrastructure planning phase. Wu et al. [14] propose a heuristic algorithm 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 they assume that the VM instances are homogeneous which means that only one virtual machine type is considered.

The Critical Path-based Iterative (CPI) [15] and complete Critical Paths (CPIS) [16] algorithms are other algorithms for solving the cloud infrastructure planning problem within the bounds of a single deadline. Based on the calculated earliest finish time and latest finish time of individual tasks, CPI identifies a complete critical path through the application workflow from start to finish and assigns the tasks in the critical path to VM services. In CPIS, a graph labelling method is applied to construct complete critical paths of the kind generated by CPI. Similar to IC-PCP, CPIS uses the VM service with best performance to find such critical paths.

Table 1: Summary of the relevant algorithms for planning cloud resources and for scheduling workflows

Algorithm	Objectives	Deadline Constraints	Workflow type	Limits
cluster and assign sub-deadline for each task [8]	cost	single	scientific	Communication time is not considered
IC-PCPD2 [3]	cost	single	scientific	Sub-deadline assignment may reduce clustering
MOHEFT [13]	cost, makespan trade-off	-	scientific	Communication time is only used in the task ranking phase but not in the planning phase
Heuristic DAG scheduling algorithm [11]	cost	-	scientific	Communication time is not considered
IC-PCP [3]	cost	single	scientific	All the tasks in the critical path assigned to the same VM
PSO [4]	cost	single	scientific	Encoding makes schedule generation process hard to determine
CPI [15]	cost	single	service	High time complexity in the path assignment
CPIS [16]	cost	single	service	High time complexity in the path assignment

Table 1 summarises the related work. Existing works about 'multiple deadline' workflow scheduling are mainly about scheduling multiple workflows with each workflow having a single global deadline. For planning infrastructures for workflows with multiple deadlines we can apply a single deadline scheduling algorithm adapted to the multiple deadline case, or a new approach can be tried. In this paper we examine both approaches in order to permit us to perform comparative analysis of our solution with existing work; the algorithm that can be most easily adapted to this situation is IC-PCP.

3. Problem Formulation

The QoS requirements of cloud applications are diverse, being based on the type and requirements of the individual application. According to the state of the art, the most widely discussed QoS requirements include deadline, budget, power consumption, reliability, security and the aggregation of these requirements [17]. For time critical applications, deadline is one of the most important QoS parameters. Even if the functional correctness of the application can be guaranteed, the violation of deadlines can still lead to application failure. Cloud infrastructures should therefore be carefully planned to maximise the likelihood of meeting the timing constraints of time critical applications. Given the standard pay-as-you-go model used in clouds, monetary cost is also an important concern for developers deploying their applications in the cloud. In this paper we confine the problem of workflow planning to that of planning a VM infrastructure to host the workflow of a time critical application that will satisfy its (multiple) deadline requirements while minimising monetary cost.

Deployment of tasks on different VM services leads to different levels of performance, resulting in different impacts on the application QoS and cost. We assume that after one task transfers its results to all its successors, the VM where the task is deployed is not released. Instead, the task will act as a persistent service waiting for more input; thus the deadline for a given task must be satisfied every time the task receives new input. This notion of persistence is common for service-oriented architectures, but is quite different from the assumptions made be many works in scientific workflow scheduling [3, 4, 18, 19]. The tasks in such workflows usually have a longer duration, with some tasks requiring days or even weeks to complete, even when deployed on a high performance infrastructure. When one task finishes its processing, the resources upon which it is deployed can then be released or taken over by other tasks. However, for most time critical applications, tasks are typically persistent, being required to remain operational until the whole application is completed. Individual invocations of tasks require a response in the scale of minutes or even (milli)seconds. Thus, it is almost impossible to respond to new input within time constraints if resources have been released and need to be reclaimed. Another assumption we make is that all the tasks in the workflow will be deployed on non-shareable VM services because it has been shown that sharing VMs impacts the performance of tasks [16].

The workflow of a cloud application can be represented as a Directed Acyclic Graph (DAG) that explicitly reveals the data dependencies between tasks [17]. In this paper we use $G = \langle V, E \rangle$ to represent the workflow of an application. $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes v that corresponds to tasks in workflow G. For each task $v \in V$, we define

the parents of v as $pred(v) = \{v' \mid v' \in V \wedge (v', v) \in E\}$. 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 service provided by the cloud provider and cannot be split into two or more sub-tasks.

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 $v \in V$. For applications with multiple initial or final tasks, it is possible to adapt their workflows by attaching dummy tasks to the start or end of the workflow with zero communication cost to the actual initial or final tasks respectively, however some adaptation of the algorithms presented in the next section will then be required.

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 optimized), C (computing optimized) and R (memory optimized) VMs as offered by Amazon EC2 [20]. In this paper we denote such VM services as basic service types. Each task in the workflow can be deployed on a instance of one VM service type. When we refer to a *VM service*, 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 is p. Deployment of tasks on different VM services will result in different performance, which can be represented by a performance matrix T:

$$T = \begin{array}{c} \\ s_1 \\ s_2 \\ \cdots \\ s_m \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & \cdots & v_n \end{array} \\ \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{22} & \cdots & t_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ t_{m1} & t_{m2} & \cdots & t_{mn} \end{bmatrix} \end{array}$$

Each element t_{ij} 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. The response time of a task depends on the type of VM where it is deployed and the input workload. The input workload for persistent tasks usually follows the same pattern within a certain period, with the number of instances of input being retrieved over the course of the application increasing with application complexity rather than the size of individual input instances themselves; we therefore assume a static value for the performance based on the worst case observed performance. Performance observations can be obtained from historical data [21] or using performance estimation methods [4]. Due to the dynamics introduced by e.g. resource sharing, network delay and VM consolidation in clouds however, the performance can vary over time even when the same task is executed with the same workload on the same VM [19].

For a workflow graph $G = \langle V, E \rangle$, 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. We can add another entity W to represent the communication cost between tasks in the workflow. The communication cost denotes the data transfer time between one task to another. We assume that the bandwidth among VMs in the same data center is always the same. Thus data transfer time is mainly determined by the size of the data being sent, which for each individual invocation of a task we assume to be mostly the same irrelevant of input. Let $W[v, v']$ return the communication cost between a task v and a task v', derived by equivalent means to those used to derive the values of a performance matrix T as described above.

For time critical applications in the cloud, performance variation in different instances of the same task on same type of VM should not be neglected, and so it may be worth in some cases injecting additional factors δ_1 and δ_2 representing performance fluctuation within VMs and between VMs respectively. If such factors are being included, then for the purposes of planning, given a performance matrix T and a communication cost function W, task v_j on a service s_i should be treated as having an actual execution cost of $t_{ij} \times (1 + \delta_1)$ and each communication link between two tasks v and v' should be treated as having an actual communication latency of $W[v, v'] \times (1 + \delta_2)$. Given a reasonably accurate description of the performance of tasks on different VM services and the communication cost between tasks in a given data center, we can determine the suitability of certain services for particular tasks. The execution of the tasks in the cloud should not violate the data dependencies of the tasks in the workflow, which means that a task can start execution only when all its predecessors have completed their processing and the data

communication between them is finished. We use Q to represent the set of deadline requirements q of the time critical application workflow, where $q = \langle v, d \rangle$ denoting that task v should complete before time d, as measured from the start of the workflow. We use $D(v) = d$ to represent the deadline of v. 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 $Q = \{\langle v_{exit}, d_1 \rangle\}$ where v_{exit} is the final task in the workflow.

Fig. 1 shows a typical application workflow with successive overlapping deadlines, interpreted as a disaster early warning system. The processing of the workflow starts from task $v_0 = v_{entry}$ and ends with task $v_{10} = v_{exit}$. Each node in the workflow represents a task, which can be deployed using VM services offered by the cloud provider. The arcs between nodes represent communication dependencies, annotated with their communication costs. The entry node is a node of the workflow DAG which indicates the onset of the processing of the persistent services represented by nodes v_1, v_2 and v_3, which are three distributed data pre-processing tasks. Task v_4 performs synchronisation and selection of pre-processed sensor data, which is forwarded to v_5 to be used in predictive simulation. The forecast result is transferred to v_6 which then passes it on to parallel disaster assessment modules v_7, v_8 and v_9. Assessments are aggregated by a final task v_{10}.

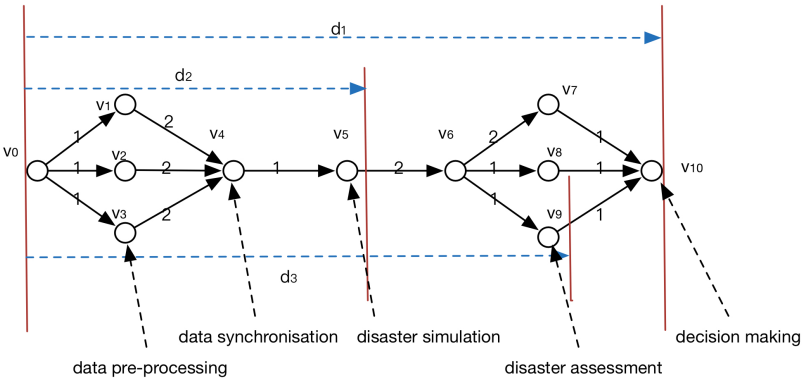

Figure 1: Example of an abstract early warning system worflow with multiple deadlines.

Fig. 1 presents three deadlines d_1, d_2 and d_3 constraining the processing time for events; a global deadline d_1, and two intermediate deadlines d_2 and d_3 imposed on simulation and disaster assessment respectively. We want to select VMs for tasks in the workflow such that all three deadlines can be met and the cost of hosting the application is minimised. Choosing VM services for tasks with better performance makes it easier to meet deadlines, but can lead to unnecessary extra cost if more modestly performing VMs can suffice. The problem thus turns into a constrained optimisation problem. Generic solutions like IC-PCP [3], CPIS [16] and CPI [15] can find solutions for meeting the global deadline d_1 by applying heuristic algorithms, but may not meet the other two deadlines depending on how they influence one another—for example if d_2 is a very loose deadline and d_1 is comparatively tight, then if we plan VMs for the workflow considering d_2 using the IC-PCP approach, d_1 might still be violated. Conversely, if d_1 is a loose deadline and d_2 is tighter, the d_2 can be violated if we use IC-PCP only considering d_1.

In principle, IC-PCP can be adapted to plan the kind of service-based workflows discussed in this paper by using the sum cost of VMs per time unit as the metric for measuring whether one assignment is cheaper than the other and forbidding multiple tasks from being assigned to the same VM instance. By changing the calculation on the latest finishing time to take into account internal deadlines, a 'minimally modified' variant of IC-PCP (which for brevity we will refer to as IC-PCP*) can plan for workflows with multiple deadlines. What we found however was that this approach still incurs unnecessary monetary cost—it is possible to drive the cost down further than such a minimal adaptation of IC-PCP permits in many cases. To illustrate this, we propose a more substantive modification of the approach provided by IC-PCP that we refer to as Multi-dEadline workflow Planning Algorithm (MEPA). In the next section, we describe our algorithm asnd its characteristics in detail, and then proceed to compare it experimentally against IC-PCP* against a wide range of randomly generated workflows.

4. Multi-dEadline workflow Planning Algorithm (MEPA)

As described in the previous section, a time critical application with multiple internal deadlines can be modelled as a set of inter-dependent tasks deployed on VM instances. We consider the assignment of $s_j|\min_{j\in(0,m]} t_{ji} \times (1+\delta_1)$ to task v_i to be the assignment of v_i that allows v_i to respond quickest to input. Conversely, the assignment of $s_j|\max_{j\in(0,m]} t_{ji} \times (1+\delta_1)$ for task v_i is the worst assignment of v_i. The 'best' assignment of all tasks in the workflow does not necessarily equate to an optimal solution however, because while the makespan of the workflow might be minimised, the monetary cost for hosting the application is not itself being optimised.

In this paper, we assume the service quality of virtual infrastructures will be guaranteed by their providers; the possible failures of virtual machines and application components are thus not specifically considered in the planning algorithm.

4.1. MEPA description

The pseudo code of MEPA is shown in Algorithm 1. MEPA uses a "compress-relax" method to solve the workflow planning problem: we "compress" by initially assigning tasks to VM services with better performance so that the makespan of the workflow is compressed within the deadline, and we "relax" to by re-assigning tasks to VMs with worse performance but lower cost (while still satisfying all deadlines).

Algorithm 1: MULTI-DEADLINE WORKFLOW PLANNING ALGORITHM (MEPA)

Input: G, T, W, Q
Output: Planned VMs
for v_i *in G* **do**
  Initialise v_i with $s_j|\min j \in (0,m] t_{ji} \times (1+\delta_1)$
  Set task v_i as unassigned
Calculate the EST of tasks in the workflow
$LFTCalculation(G, v_{exit}, T, W, Q)$ *(Algorithm 2 below)*
while *there exist unassigned tasks* v_i **do**
  $ConstructPCP(G, TW, Q, v_i)$ *(Algorithm 3)*
  Assign tasks in PCP with GAPA *(see Section 4.2)*
  Set the tasks in the PCP as assigned
  Update the EST, EFT and LFT of the tasks in the workflow
return planned VMs.

Initially MEPA assigns each task in the workflow with the best performing VM to guarantee a solution if one exists—if all deadlines cannot be met using the best VM services available, then the application developer has to turn to the cloud provider to obtain VMs with even better performance or else amend their QoS requirements. Based on the initial "compressed" assignment, we then calculate the *Earliest Start Time* (EST), *Earliest Finish Time* (EFT) and *Latest Finish Time* (LFT) based on the data dependencies between tasks (and the corresponding communication costs).

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[3]:

$$EST(v_{entry}) = 0$$

$$EST(v_i) = \max_{v_p \in pred(v_i)} \{EST(v_p) + T[A(v_p), v_p] \times (1+\delta_1) + W[v_p, v_i] \times (1+\delta_2)\}$$

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

$$EFT(v_i) = EST(v_i) + T[A(v_i), v_i] \times (1+\delta_1)$$

We take the global deadline of the workflow and assign it to the LFT of the exit node, and then we iteratively assign the internal deadlines of tasks; Algorithm 2 shows how the LFT of each task is calculated. The algorithm applies a

recursive function which takes one task in the workflow as input and backtracks through all its parents to assign LFTs. We use $D(v_p)$ to represent the deadline of task v_p. If tasks have user-defined deadlines, then we compare whether each deadline is earlier than the calculated LFT. If the deadline is earlier than the LFT, it means that the deadline has stricter requirement, and so we simply assign that deadline as the task's revised LFT. Otherwise, we take the calculated LFT as the deadline of each task because if an assignment of a task violates that LFT, then it will be impossible to meet later deadlines with the services available. Formally, the LFT is defined as:

$$LFT(v_i) = \begin{cases} \min_{v_c \in succ(v_i)}\{LFT(v_c) - T[A(v_c), v_c] \times (1+\delta_1) - W[v_i, v_c] \times (1+\delta_2)\} & \langle v_i, \cdot \rangle \notin Q \\ \max\{D(v_i), \min_{v_c \in succ(v_i)}\{LFT(v_c) - T[A(v_c), v_c] \times (1+\delta_1) - W[v_i, v_c] \times (1+\delta_2)\}\} & \langle v_i, \cdot \rangle \in Q \end{cases}$$

The LFT is calculated starting from the exit task, which is initially assigned with the global deadline of the entire workflow. In the workflow, the user may or may not specify the global deadline. Therefore, encountering such situation, we assign a 'large' deadline so that the assignment of the workflow even with cheapest VM can meet the global deadline. The 'large' deadline is set as the sum of the worst case response time of each task plus all the communication time. $W(e)$ represents the communication time the source and destination of e. Formally, the exit task's LFT is calculated as:

$$LFT(v_{exit}) = \begin{cases} D(v_{exit}) & \langle v_{exit}, \cdot \rangle \in Q \\ \sum_{i=1}^{n} T[m,i] \times (1+\delta_1) + \sum_{e \in E} W(e) \times (1+\delta_2) & \langle v_{exit}, \cdot \rangle \notin Q \end{cases}$$

Whenever the EFT of a task is greater than its LFT, the currently available services cannot satisfy the time constraints of the workflow because the task will always finish late.

Algorithm 2: LFTCALCULATION

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

```
if pred(v) = ∅ then
    return
else
    for v_p ∈ pred(v) do
        lft = LFT(v) − T[A(v), v] × (1 + δ1) − W[v_p, v] × (1 + δ2)
        if lft < LFT(v_p) then
            LFT(v_p) = lft if ⟨v_p, ·⟩ ∈ Q ∧ D(v_p) < LFT(v_p) then
                LFT(v_p) = D(v_p)
            if LFT(v_p) < EFT(v_p) then
                Report that the current cloud provider cannot guarantee the deadline of the workflow
            return LFTCalculation(G, v_p, T, W, Q)
```

In this paper we construct the partial critical path using the same method as for IC-PCP. However, to enhance performance, the IC-PCP will only allow each node be assigned once. Thus, the partial critical path means the longest path between two unassigned nodes.

A critical parent of v is defined as the unassigned parent of v_i with the latest arrival time at it [3]. Formally, the critical parent is defined as:

$$CParent(v) = \{v' \mid v' \in pred(v) \wedge max\{EFT(v') + W[v', v] \times (1+\delta_2)\}\}$$

A partial critical path of a task v can be formally defined as [3]:

$$PCP(v) = \begin{cases} empty & CParent(v) = \emptyset \\ CParent(v) + PCP(CParent(v)) & CParent(v) \neq \emptyset \end{cases}$$

Pseudo-code of the partial critical path construction algorithm is show in Algorithm 3. After initialisation, the partial critical path is constructed starting from a node v_k, adding the parent of v_k with the latest arrival time to the partial critical path until an assigned node is reached.

The main difference between IC-PCP and solutions like CPI and CPIS is that IC-PCP defines partial critical paths while CPI and CPIS apply a Graph Labelling Method (GLM) to construct the complete critical path. The concept of GLM is quite similar to the concept of Critical Parent in IC-PCP which finds the parent that has the latest arrival time on the current task. In IC-PCP, each task in the workflow is assigned once. However, in CPI and CPIS, the task can be assigned more than one time because the complete critical paths constructed each iteration may have some intersecting tasks. The constructed partial critical path is assigned with a Genetic Algorithm based Planning Algorithm (GAPA) which will be discussed later in this chapter. After assignment, the tasks in the PCP are tagged as assigned and the EST, EFT and LFT of the tasks in the workflow are updated. This process will continue until all the tasks in the workflow are assigned.

Algorithm 3: PARTIAL CRITICAL PATH CONSTRUCTION ALGORITHM

Input: G, T, W, Q, v_k
Output: A partial critical path

PCP = (v_k)
while *pred(v_k) has unassigned members* **do**
    find v_i that has $\max_{v_i \in pred(v_k)} EFT(v_i) + W[v_i, v_k] \times (1 + \delta_2)$
    append v_i to PCP
    $v_k = v_i$
return PCP

4.2. GAPA description

We now describe how we approach the partial critical path assignment problem. A partial critical path can be seen as a sequential workflow where each task in the workflow has only one predecessor and one successor, except for the entry and exit tasks. We assume that the tasks $v_1, v_2, \ldots, v_k$ in the partial critical path $PCP(v_k)$ with length k are ordered based on the data dependencies in the workflow, which means that v_j will not execute before v_i finishes if $1 \leq i < j \leq k$. Now assume the cloud provider offers m different VM services. If each task in the critical path can be assigned to any type of VM service, there are m^k choices for assigning the path in total. When the length of the partial critical path or the number VM service types increase, it becomes increasingly prohibitive to iterate over all possible solutions. We can consider the calculation of the overall deadline of a partial critical path as the capacity of a knapsack and the other deadlines within the critical path as constraints and formulate the problem as a Multiple Level-constrained Multiple Choices Knapsack Problem (MLMCKP). Traditional Multiple Choice Knapsack Problems (MCKPs) consider a set of item classes and try to select an item for each class to achieve objectives like value maximisation within the capacity of the knapsack [22]. In MCKP, there is only one constraint, which is the size of the bag. However in MLMCKP there are additional internal time constraints. If we consider the overall deadline of the workflow as the size of the knapsack, the internal time constraints can be seen as the size constraint of additional knapsacks inside the 'global' knapsack. MCKP differs quite a lot from the traditional Binary Knapsack Problem (BKP), which only decides whether or not to put an item into the knapsack, having a much larger searching space than BKP. Based on the description above, and given a workflow graph $G = \langle V, E \rangle$, a performance matrix T and a communication cost function W, we formulate the MLMCKP problem as:

$$\underset{x}{\text{minimize}} \sum_{i=1}^{k} \sum_{j=1}^{m} p_j \times x_{ji}$$

$$\text{subject to } \forall k' \in [1,k], EST(v_1) + \sum_{i=1}^{k'} \sum_{j=1}^{m} t_{ji} \times (1+\delta_1) \times x_{ji} + \sum_{i=1}^{k'-1} W[v_i, v_{i+1}] \times (1+\delta_2) < LFT(v_{k'})$$

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

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

The objective of MLMCKP is to minimise the cost of the partial critical path within the constraints of the LFT of each task. In this paper we use the sum of VM instance price per time unit to represent the cost because the money users need to pay depends on it and the lifetime of the VM reservation which is specified by the user. We do not consider the internal deadlines because these time constraints have already been subsumed in the LFTs already calculated. We can use the summary price of all planned VMs because all the tasks in the workflow are persistent services that last the duration of the workflow; therefore the total cost of executing the workflow is a constant factor of the combined summary price of VMs. p_j represents the price per time unit of service s_j. This model can also be applied to scientific workflows by changing the objective to be the aggregation of price per unit times the duration of each instance. We take the same assumption as IC-PCP that after assignment each task in the critical path should not violate its LFT. When IC-PCP assigns a VM to a path, it chooses "a new instance of the cheapest service which can finish each task of the critical path before its LFT" [3]. Because IC-PCP initially assigns to each task in the partial critical path the VM with the best performance, the LFT calculated after initialisation should not be violated because if the VM service with the best performance cannot meet the LFT of a task, then the cloud resources cannot meet the timing constraints of the workflow. $EST(v_1)$ represents the earliest start time of the first task in the partial critical path. The $LFT(v_{k'})$ represents the Latest Finish Time of task $v_{k'}$. $\sum_{j=1}^{m} x_{ji} = 1$ serves to assert that only one service will be selected for each task.

As an example, we revisit the workflow shown in Fig. 1. Assume that the cloud provider offers three different types of VM service, and that the response time of the tasks in the workflow of Fig. 1 for each of the three different VM types is described by T below. The prices per time unit of the VM services s_1, s_2, s_3 are $5, 2, 1$ respectively. We set the time constraints in the workflow d_1 as 60, d_2 as 26 and d_3 as 40.

$$T = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 2 & 4 & 5 & 3 & 7 & 6 & 5 & 5 & 7 & 2 \\ 2 & 4 & 5 & 7 & 6 & 8 & 9 & 7 & 6 & 8 & 5 \\ 3 & 5 & 8 & 8 & 8 & 10 & 12 & 10 & 8 & 10 & 7 \end{array} \right] \end{array}$$

In IC-PCP's initialisation phase, the EST, EFT and LFT of all the tasks are calculated by assigning the VM service with the best performance for each task. We therefore initially assign s_1 to all the tasks in the workflow. Using Algorithm 3, we obtain a critical path of the workflow consisting of seven tasks $\{v_0, v_3, v_4, v_5, v_6, v_9, v_{10}\}$. We calculate the LFT of these tasks all along the critical path based on Algorithm 2, equalling 4, 11, 16, 24, 32, 40 and 60 respectively. According to the path assignment algorithm in IC-PCP, relying solely on instances of service s_1 for all tasks in the partial critical path will meet the LFT of the tasks with a total cost of 35. However we can find an alternative assignment of service types to tasks $\{s_3, s_2, s_1, s_2, s_1, s_1, s_3\}$ with a total cost of 21 that can still meet all internal deadlines. For the path assignment algorithm, IC-PCP assigns all the tasks in the critical path to the same VM type. Such assignment can lead to failure of some deadlines, or at the very least introduce unnecessary extra cost as shown in the example above. We therefore formulate the problem of path assignment as a MLMCKP problem and propose a genetic algorithm based approach to solve the problem of identifying good path assignments. Evolutionary algorithms have been proven to perform well when dealing with problems that have complex objectives and constraints [23]. We therefore use a genetic algorithm to select the best mapping of VM services to tasks, referring to our specific implementation as GAPA (Genetic Algorithm based Planning Algorithm).

In GAPA, we maintain a population in each generation. The population is a set of candidate solutions (namely chromosomes). We encode each type of VM provided by the cloud provider as a value between 1 and m. When a partial critical path of length k is identified, we encode the tasks in the path $v_1, v_2, \ldots, v_k$ as an individual with a chromosome $y_1, y_2, \ldots, y_k$ where $1 \leq y_i \leq m$. Each element in the chromosome denotes that v_i is planned with a VM of type s_{y_i}. In the initialisation phase of the genetic algorithm, a population consisting of a certain number of individuals is initialised for a critical path. The chromosomes of all the individuals in the initial population are initialised randomly. We generate a random value ranging from 1 to m for each position of a chromosome and all the individuals in the population have the same length as the partial critical path. Apart from the randomly generated individuals, we also generate individuals with chromosomes that assign the whole partial critical path to the same type of VM, resulting in m additional assignments.

The objective of our algorithm is to minimise monetary cost; thus cost is part of the fitness function. The cost is calculated by summing up the prices per time unit of VM services $\sum_{i=1}^{k} p_{y_i}$. The constraints on LFT still apply, so the randomly generated individuals are not always acceptable solutions. When any violation of the critical path LFT happens, a heavy penalty is added so that such unacceptable assignments are easily eliminated in later generations. The fitness of a solution is thus the inverse of the sum of the monetary cost and any accumulated penalties:

$$1.0/(\sum_{i=1}^{k} p_{y_i} + c \times \sum_{i=1}^{k} penalty_i)$$

$$penalty_i = \begin{cases} 0 & \text{LFT and deadline of task i are not violated} \\ 1 & \text{LFT or deadline of task i is violated} \end{cases}$$

In evolutionary algorithms, the genetic operators (crossover, mutation and selection) are key to the performance of a genetic algorithm. The crossover operator of a genetic algorithm serves to generate the offsprings of the population through the crossover of some parts of the parents' chromosomes. In GAPA, any two individuals in the previous population are mated with certain probability and the chromosomes are intersected by exchanging two randomly chosen points. The mutation operator ensures population diversity and stops possible paths to solutions from being prematurely discarded. After crossover, the chromosome of each individual is mutated with a certain probability. When one or more places in a chromosome are chosen as the mutation position, the number in the position is randomly regenerated with a different type of VM. We keep the same size of population for each generation. So the same number of chromosomes are selected in each generation by choosing individuals from the combination of the previous generation and a newly generated generation with higher fitness values, eliminating solutions with lower fitness value. In this way, the unfeasible solutions can be eliminated in the new generation.

Usually the genetic algorithm stops once a specific number of generations have passed, or a feasible solution is found that can meet the requirements of the user. For MLMCKP, a feasible solution is an assignment of VMs to each task that can meet the LFT or deadline of all tasks. Assigning the VM service with the best performance for each task is a simple solution, but can incur a high monetary cost. As MCKP has already been proven to be NP-hard, it is difficult to determine when an optimal solution has been found, so we limit the number of generations produced based on the length of the partial critical path and select the best solution in the final generation. In our experiments, we set the generation limit to $a \times len(pcp)$, where a is a constant factor that can be tuned by the user; smaller for faster processing, larger for better solution quality.

5. Experiments

Our implementation of MEPA is based on Python 2.7.10. We use NetworkX (version 1.10) [24] to manage the workflow and PyDOT2 (version 1.0.33) [25] to parse the graphs generated by GGen [26]. NetworkX is a powerful Python library for manipulating complex networks. GGen is an open source random graph generator integrating several different random graph generating algorithms. The generated random DAGs are represented in DOT, which is a plain text graph description language. DEAP (Distributed Evolutionary Algorithms in Python) [27] is a framework for experimenting with evolutionary algorithms such as genetic algorithms and particle swarm optimisation. In this paper we use DEAP as the underlying framework for implementing GAPA. We conduct our experiment on the Distributed ASCI Supercomputer 5 (DAS-5) [28].

5.1. Workload generation

To investigate the behavior of our algorithm, we use the graph generator GGen [26] to generate random workflow typologies with different time constraints. Specifically, we apply 'fan-in/fan-out' methods to generate DAGs, which are widely used in random graph generation. This graph generation method takes three parameters: the number of vertices, the maximum in-degree of each node and the maximum out-degree of each node. This kind of graph generation method will generate a graph topology with all tasks' in-degrees and out-degrees within the chosen upper bounds. If the in-degree and out-degree is set to be one, then the DAG becomes a sequential graph. In order to test how our solutions perform on different scales of graph, we set the number of vertices in the workflows to range from 2^0 to 2^8. The in-degree and out-degree are used to generate DAGs with different shapes and we set the maximum in-degree and out-degree to range from 1 to 5 and 1 to 4 respectively.

For each DAG we need to generate an execution profile. The execution profile includes the performance of tasks on different VM services as well as the communication costs between tasks, which ranges from 1 to 200. In many time critical applications, the performance of tasks on different services varies for each task. The response time of tasks may be less than or greater than the communication cost depending on the nature of computation being performed and the quality of the network. So in order to make the data more realistic, we should randomly generate response times for tasks running on different VM services that can both exceed and be significantly less than communication times. For each task we first generate the execution cost of the task on the 'best' VM service, randomly selecting a response time between 1 and half of the communication cost upper bound. The execution costs of the task on the other 'lesser' services are generated iteratively by increasing the previously generated cost by a randomised proportion. In order to simulate better different kinds of real world application, our performance generation method ensures that the performance of each task on different VM types can be substantially larger than the communication cost or much smaller, ensuring greater diversity in the workflows generated and removing any implicit assumption about the relative cost of computation versus communication.

The time constraints attached to a workflow are also randomly generated. The number of time constraints are set with a proportion to the scale of the workflow. Specifically, we set the number of time constraints per workflow to be $\lceil 0.1 \times |V| \rceil$, where $|V|$ is the number of tasks in the workflow. We then randomly select $\lceil 0.1 \times |V| \rceil$ tasks from the workflow (with the exception of the last task, which is always the final task in the workflow), and for each task we attach a random deadline based on the critical path calculation performed during workflow generation, limiting each deadline's range based on best and worst performing VM services so as to ensure no 'impossible' (or far too easy) deadlines are set. The final task will always receive a deadline, which will serve as the global deadline for the entire application. All datasets generated for the experiments in this paper are available online [1].

5.2. Comparison of path assignment with IC-PCP and GAPA

The partial critical path can be seen as a sequential workflow, each task of which has only one predecessor and successor except for the entry and exit tasks. We therefore use a set of sequential workflows to test the performance of GAPA. We set the scale of the sequential workflow ranging from 10 to 100 and the proportion of deadlines is set to be 0.1. The performance matrix of the tasks in the critical path is generated randomly as described in Section 5. Considering the performance fluctuation, for the generated performance profile, we set the task performance fluctuation rate to be 10 percent and communication fluctuation rate to be 15 percent. The multiple time constraints of the workflow are generated as described in Section 5. We set the final generation in GAPA to be equal to the length of the partial critical path, which means that $a = 1.0$. In GAPA, we set the mutation rate to be 0.05 and population size to be 300. We assume the cloud offers three different types of VM services and set the price for each service as 5, 2 and 1, which is the same as used in [3].

In IC-PCP, the whole partial critical path is assigned with the same VM type. Thus, for each workflow, there are three path assigning choices, assigning all the tasks to s_1, s_2 or s_3. We compare the cost with GAPA and the cost with IC-PCP by assigning all the tasks to the VM service with best performance. We do this because other VM service selections usually violate one or more deadlines; based on our experiment involving 90 partial critical paths of incrementally increasing length, assigning the second best type of VM leads to a valid solution only 4 times in 90.

[1] https://github.com/WorkflowPlanning/workload

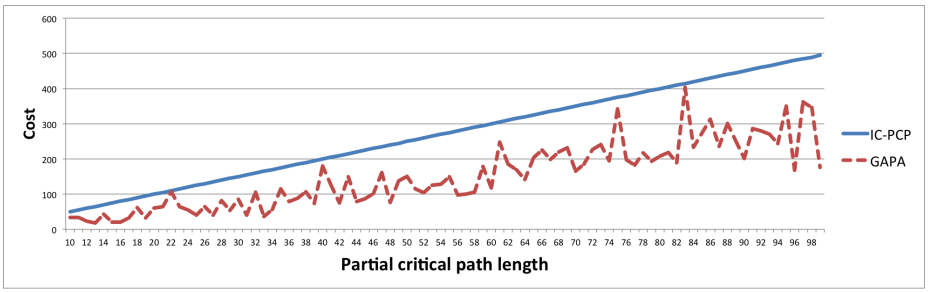

Figure 2: Cost comparison of path assignment in IC-PCP and GAPA

Experimental results are shown in Fig. 2. The cost of assigning the best performing VM service to all tasks follows a linear incremental trend because the assigned VMs are of the same type. The result gotten from GAPA varies a lot because the deadline is randomly generated and the solution obtained from GAPA may not assign all the tasks to the same VM type at different path lengths. We can see that the assignment with GAPA appears to perform consistently better than when simply assigning all tasks on the VM with the best performance. We also find that GAPA can save up to almost two thirds of the IC-PCP cost in this experiment.

5.3. Comparison of IC-PCP, CPI and MEPA with a single deadline

MEPA can plan virtual infrastructures for applications with multiple deadlines, but to compare it with existing planning approaches that only support a single global deadline, we have conducted experiments on the dataset described in Section 5, applying a single deadline and comparing the results of MEPA with IC-PCP and CPI.

Fig. 3, Fig. 4 and Fig. 5 show the planned virtual infrastructure cost of solutions produced by MEPA, IC-PCP and CPI with test workflows with 16, 32 and 64 tasks respectively, in each case varying both in-degree and out-degree (identified along the x-axis with the in-degree above the out-degree) to ascertain how connectivity influences results. From the figures we can see that MEPA generally leads to less expensive VM assignments than IC-PCP. MEPA can even save around 66 percent of cost compared with IC-PCP in some cases. Although from the results we can see that MEPA and CPI give solutions with similar costs, the time complexity of the CPI is $\mathcal{O}(N^3D^2M)$ due to the assignment of the path with dynamic programming to find the Pareto assignment [15], making it hard to scale when the deadline of the workflow is very large. N represents the number of nodes. M represents the number of services and D represents the global deadline. In our solution, there is no such bottleneck with the scale of the deadline.

Moreover, we can see that for a workflow with the same number of nodes, the cost of MEPA and IC-PCP, CPI are quite close. The reason for this is that when the in/out degree increase, the DAG will become "wider", making the length of the critical path become shorter. For a "loose" deadline, the path assignment of MEPA and IC-PCP can lead to similar solutions, leading to similar total cost. Moreover, it is not hard to see that when the scale of the workflow increases, the differentiation between MEPA and IC-PCP will become more significant.

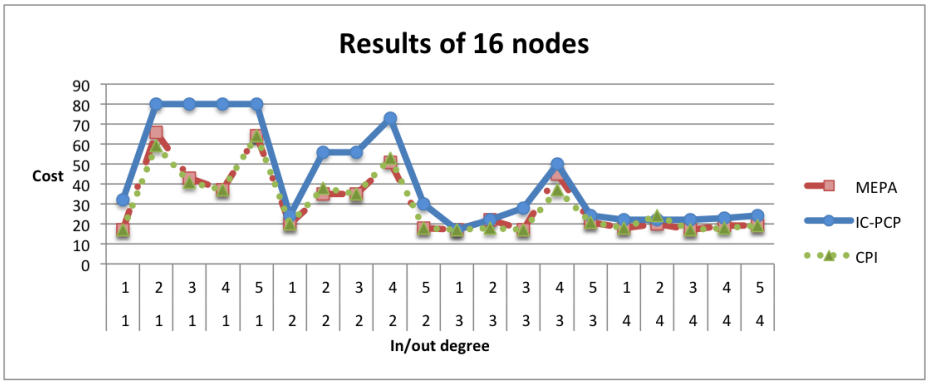

Figure 3: Results of 16 nodes with IC-PCP, CPI and MEPA of single deadline

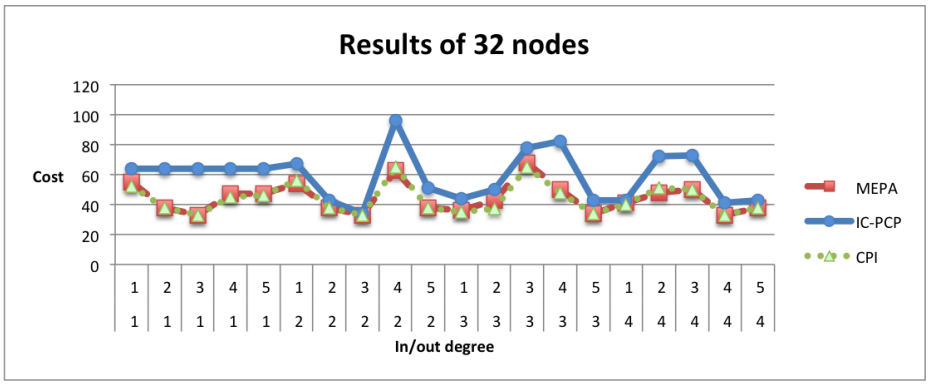

Figure 4: Results of 32 nodes with IC-PCP, CPI and MEPA of single deadline

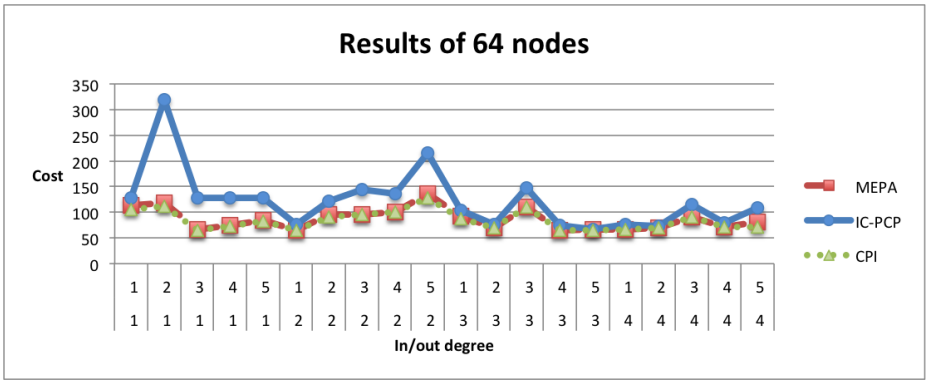

Figure 5: Results of 64 nodes with IC-PCP, CPI and MEPA of single deadline

5.4. Comparison of IC-PCP and MEPA with multiple deadlines*

In this part we take the workload described in Section 5 and feed the workload into both IC-PCP* (the minimal modification of IC-PCP for multiple deadline workflows described in section 3) and MEPA. Fig. 6, Fig. 7 and Fig. 8 compare the results of IC-PCP* and MEPA with workflows of size of 16, 32 and 64 with different in-degrees and out-degrees (identified along the x-axis with the in-degree above the out-degree in all figures). We can see from the results that MEPA is able to give cheaper solutions than IC-PCP*. When the scale of the workflow increases, the differentiation between the results of MEPA and IC-PCP* will become more significant. With the increase of in/out degree, the result of MEPA and IC-PCP* tend to become similar, but the similarity is reached for larger in/out degree ratios when the number of tasks in the workflow increases. This is because the length of the partial critical path will become shorter when the in/out degree increases. So the internal deadlines in each partial critical path can be less, making the planned results more similar.

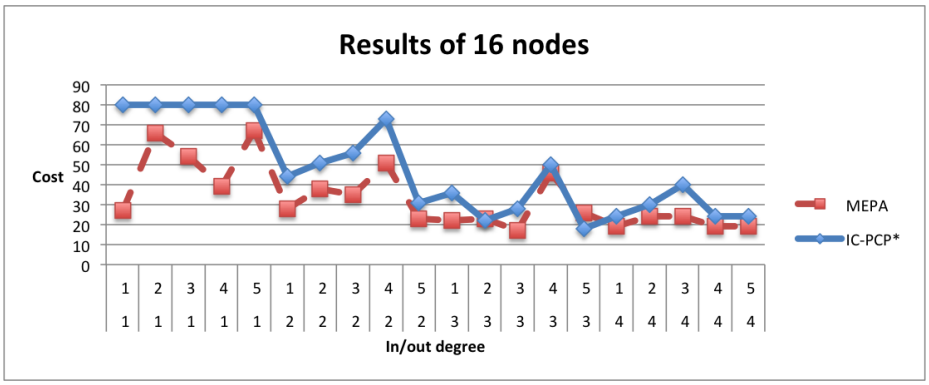

Figure 6: Results of 16 nodes with IC-PCP* and MEPA of multiple deadlines

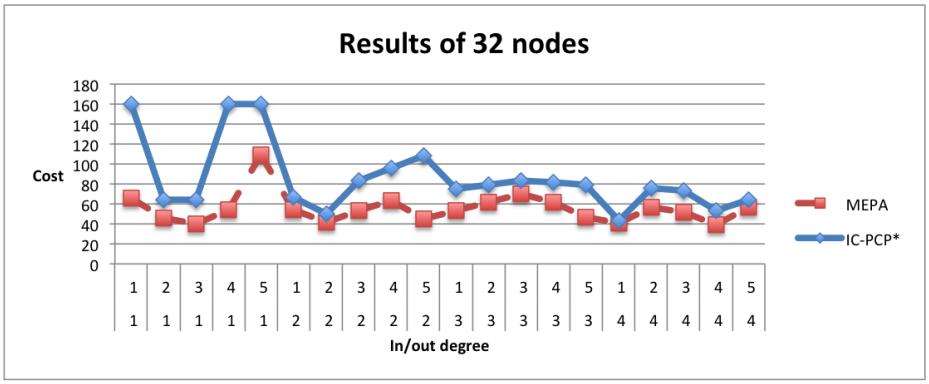

Figure 7: Results of 32 nodes with IC-PCP* and MEPA of multiple deadlines

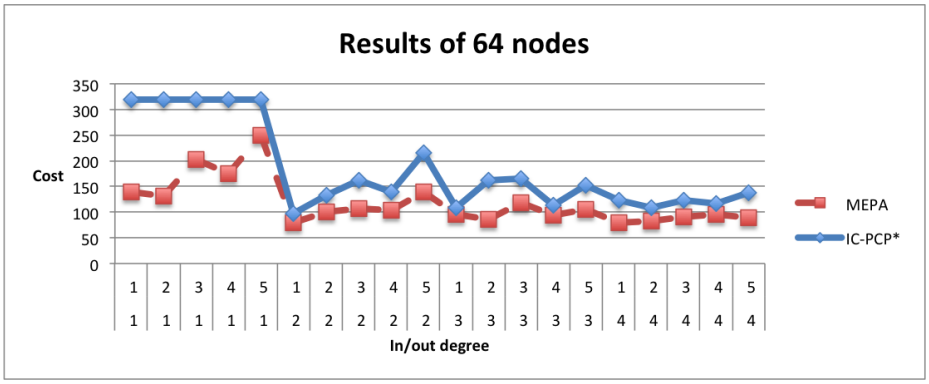

Figure 8: Results of 64 nodes with IC-PCP* and MEPA of multiple deadlines

6. Conclusion and Future works

In this paper we investigated the problem of planning cloud virtual infrastructure for time critical applications that focused on guaranteeing multiple overlapping deadlines imposed on tasks in the application workflow while minimising monetary cost. Based on the comparison with existing work, we proposed a workflow planning approach called MultiDeadline workflow Planning Algorithm (MEPA). Our algorithm works by first "compressing" the workflow by

assigning the best VM for all the tasks and then "relaxing" it by constructing partial critical paths based on IC-PCP in order to identify where cheaper VMs can be used without violating time constraints. We formulate the path assignment problem as a kind of multiple choice knapsack problem and apply a genetic algorithm to find possible solutions. To test the effectiveness of MEPA, we randomly generated a wide range of application workflow topologies with different characteristics, assigned deadlines, and generated a range of performance profiles to match those topologies. We then compared the performance of our solution with a version of IC-PCP adapted for the particular assumptions of our problem, and also compared performance in single deadline scenarios with the original IC-PCP algorithm. The experimental results show that our algorithm can provide better solutions (in terms of minimising unnecessary monetary costs) for the planning of workflows with multiple deadlines for provisioning in cloud environments.

In our future work, we will investigate the dynamic aspects of time critical applications—considering how virtual infrastructures can be planned for re-configurable and otherwise dynamic workflows, including possible failures of virtual infrastructures and application components during workflow execution. Moreover, the infrastructure planning problem has not been widely discussed from the networking perspective, so we intend to study Software-Defined Networking technologies to better enable *network-aware* workflow planning.

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). The research is also partially funded by the COMMIT project.

References

[1] Z. Zheng, Y. Zhang, M. R. Lyu, Investigating qos of real-world web services, Services Computing, IEEE Transactions on 7 (1) (2014) 32–39.
[2] Y. Sun, J. White, S. Eade, A model-based system to automate cloud resource allocation and optimization, in: Model-Driven Engineering Languages and Systems, Springer, 2014, pp. 18–34.
[3] S. Abrishami, M. Naghibzadeh, D. H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Generation Computer Systems 29 (1) (2013) 158–169.
[4] M. A. Rodriguez, R. Buyya, Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds, Cloud Computing, IEEE Transactions on 2 (2) (2014) 222–235.
[5] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-Science: An overview of workflow system features and capabilities, Future Generation Computer Systems 25 (5) (2009) 528–540.
[6] V. V. Krzhizhanovskaya, G. Shirshov, N. Melnikova, R. G. Belleman, F. Rusadi, B. Broekhuijsen, B. Gouldby, J. Lhomme, B. Balis, M. Bubak, et al., Flood early warning system: design, implementation and computational modules, Procedia Computer Science 4 (2011) 106–115.
[7] S. S. Kosukhin, S. V. Kovalchuk, A. V. Boukhanovsky, Cloud technology for forecasting accuracy evaluation of extreme metocean events, Procedia Computer Science 51 (2015) 2933–2937.
[8] J. Yu, R. Buyya, C. K. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: e-Science and Grid Computing, 2005. First International Conference on, IEEE, 2005, pp. 8–pp.
[9] T. Yang, A. Gerasoulis, Dsc: Scheduling parallel tasks on an unbounded number of processors, Parallel and Distributed Systems, IEEE Transactions on 5 (9) (1994) 951–967.
[10] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, Parallel and Distributed Systems, IEEE Transactions on 4 (6) (1993) 686–701.
[11] M. W. Convolbo, J. Chou, Cost-aware dag scheduling algorithms for minimizing execution cost on cloud resources, The Journal of Supercomputing 72 (3) (2016) 985–1012.
[12] L.-C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of dag scheduling heuristics, in: Grid Computing, Springer, 2008, pp. 73–84.
[13] J. J. Durillo, R. Prodan, Multi-objective workflow scheduling in amazon ec2, Cluster Computing 17 (2) (2014) 169–189.
[14] H. Wu, X. Hua, Z. Li, S. Ren, Resource and instance hour minimization for deadline constrained dag applications using computer clouds.
[15] Z. Cai, X. Li, J. N. Gupta, Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing, in: Service-Oriented Computing, Springer, 2013, pp. 207–221.
[16] Z. Cai, X. Li, J. N. Gupta, Heuristics for provisioning services to workflows in xaas clouds.
[17] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, The Journal of Supercomputing 71 (9) (2015) 3373–3418.
[18] R. N. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication, Parallel and Distributed Systems, IEEE Transactions on 25 (7) (2014) 1787–1796.
[19] D. Poola, S. K. Garg, R. Buyya, Y. Yang, K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in: Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on, IEEE, 2014, pp. 858–865.

[20] Amazon ec2 product details, accessed: 2016-3-29.
URL https://aws.amazon.com/ec2/details/
[21] W. Smith, I. Foster, V. Taylor, Predicting application run times using historical information, in: Job Scheduling Strategies for Parallel Processing, Springer, 1998, pp. 122–142.
[22] R. D. Armstrong, D. S. Kung, P. Sinha, A. A. Zoltners, A computational study of a multiple-choice knapsack algorithm, ACM Transactions on Mathematical Software (TOMS) 9 (2) (1983) 184–198.
[23] C.-W. Tsai, J. J. Rodrigues, Metaheuristic scheduling for cloud: A survey, Systems Journal, IEEE 8 (1) (2014) 279–291.
[24] Networkx, accessed: 2016-5-13.
URL https://networkx.github.io/
[25] Pydot2, accessed: 2016-5-13.
URL https://pypi.python.org/pypi/pydot2/1.0.33
[26] 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.
[27] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary algorithms made easy, Journal of Machine Learning Research 13 (2012) 2171–2175.
[28] Das-5, accessed: 2016-5-13.
URL http://www.cs.vu.nl/das5/