2017 IEEE 10th International Conference on Cloud Computing

Using Integer Programming for Workflow Scheduling in the Cloud

Yi Wang
LinkedIn
Mountain View, CA, USA
Email: yi2000us@gmail.com

Ye Xia and Shigang Chen
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA
Email: {yx1, sgchen}@cise.ufl.edu

***Abstract*—We study a fundamental problem of how to schedule complex workflows in the cloud for applications such as data analytics. One of the main challenges is that such workflow scheduling problems involve many constraints, requirements and varied objectives and it is extremely difficult to find high-quality solutions. To meet the challenge, we explore using mixed integer programming (MIP) to formulate and solve complex workflow scheduling problems. To illustrate the MIP-based method, we formulate three related workflow scheduling problems in MIP. They are fairly generic, comprehensive and are expected to be useful for a wide range of workflow scheduling scenarios. Using results from numerical experiments, we demonstrate that, for problems up to certain size, the MIP approach is entirely applicable and more advantageous over heuristic algorithms.**

***Keywords*-workflow scheduling; precedence constraints; cloud computing; mixed integer programming**

I. Introduction

There is an increasing trend that the computation cloud is used for complex workflows, such as scientific computing workflows [1], [2] and big-data analytics [3], [4]. The current workflow management systems in the cloud are becoming inadequate for the growing diversity and sophistication of complex workflows. This has resulted in long job latency, violated customer service level agreement (SLA), wasted cloud resources, and poor return on investment.

This paper investigates a fundamental problem regarding how to schedule complex workflows in the cloud. The workflows envisioned in this paper contain complex constraints and feature requirements, as well as complex objectives. Each workflow contains a set of tasks. The tasks are dependent on each other, for instance, through the data they generate and consume. The task/data dependency leads to *precedence constraints*, meaning a task cannot start its execution until the tasks that it depends on have finished and the data that it needs has arrived. In addition, workflows usually have other timing constraints such as ready times and deadlines, and the tasks have their minimum resource requirements. The workflows will be executed on a set of computing resources in the cloud, such as a collection of virtual machines (VMs). There are costs associated with operating or leasing these resources. A generic workflow scheduling problem is to decide when and where to execute different tasks of all the workflows subject to the aforementioned constraints. The problem can be formulated as an optimization problem, where a typical objective is to minimize the operating cost of the computing resources or the payment needed to use the resources. A more general objective is to maximize the difference between the gain from finishing the workflows and the cost/payment.

One of the main challenges is that such workflow scheduling problems are so tremendously complex that it is extremely difficult to find high-quality solutions by heuristic reasoning alone. In fact, it is even difficult to find a feasible solution, i.e., a solution that satisfies all the constraints posed by a problem. There is a set of studies on simpler versions of the problems and they mostly focus on heuristic algorithms, which often generate infeasible solutions, such as solutions that exceed the workflow deadlines [5]–[7]. To meet the challenge, in this paper, we explore using mixed integer programming (MIP) [8] to formulate and solve complex workflow scheduling problems. Prior research in this area has not used the MIP framework much. The MIP approach allows us to describe workflow scheduling problems precisely – capturing all the complex constraints, requirements and objectives – and after that, solve the problems optimally, thus providing the best solutions to the problems. By applying standard MIP algorithms (e.g., branch-and-bound [8]), the algorithm development time can be minimized.

We next summarize our main contributions. First, we formulated three related workflow scheduling problems in MIP. These problems are fairly generic and are expected to be useful for a wide range of workflow scheduling scenarios. They are more comprehensive than those in prior studies in that

- our formulations incorporate arbitrary precedence constraints among tasks (as apposed to the series-parallel dependency, such as in MapReduce jobs), data transfer delay, deadlines and ready times, and heterogeneous resources;
- for the optimization objective, we consider both cost minimization and profit maximization; with respect to monetary cost, we consider both long-term lease and pay-as-you-go payment models for cloud resources;
- with respect to SLA involving timing, we consider both

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.26

hard and soft deadlines.

Second, since formulating workflow problems in MIP requires non-trivial effort, we dedicate the main part of the paper to describing the details about how to formulate each set of constraints, as well as the objectives. These formulation examples demonstrate the capabilities of MIP in capturing very complex workflow scheduling situations. The techniques used there can be useful for other variants of the workflow scheduling problem. Third, we conducted numerical experiments to evaluate the effectiveness of the MIP approach and its advantages over heuristic algorithms. The results show that a well-known heuristic algorithm cannot even find feasible solutions in many cases due to the complexity of the problems, whereas the MIP solutions are not only feasible but optimal.

We next briefly review related literature. The studies closest to ours only focus on much restricted versions of the workflow scheduling problem [1], [2], [4]–[7], [9]–[16]. For instance, the task dependency is less general, e.g., consisting of parallel tasks or series-parallel tasks (such as MapReduce tasks), the processing engines are homogeneous in types, or data transfer time is not considered. Even these restricted versions are very difficult solve. Limited attempt has been made to find optimal solutions, and MIP algorithms are rarely used. Nearly all previous efforts are on developing heuristic algorithms. Because of the problem complexity, there seem to be no known structures useful for developing good heuristics. Existing heuristic algorithms appear to be doing something distinctively sub-optimal. For instance, in dealing with the deadline of a workflow, one class of approach is to divide the time budget for an entire workflow into separate time budget for each of the tasks; other algorithms simply allow the deadlines to be violated. The performance gap between the heuristic algorithms and optimal algorithms is likely to be large. But, the gap is unknown because the problems are never solved optimally.

The rest of the paper is organized as follows. In Section II, we formulate a min-cost workflow scheduling problem with hard deadline guarantee. In Section III-A, we give two more advanced formulations: One has profit maximization as the objective and it allows soft deadlines, and the other adopts the pay-as-you-go payment model. In Section IV, we present numerical results to demonstrate the effectiveness of the MIP approach. We draw conclusions in Section V.

II. Min-Cost with Deadline Guarantee

A. Overview

A set of workflows is to be scheduled in the cloud. A workflow consists of a set of computation tasks. The tasks of a workflow are subject to precedence constraints among themselves; that is, some tasks depend on other tasks in the same workflow and the dependency puts timing constraints on when a task can start execution. In most cases, the precedence constraints are due to data dependency, where a task reads and manipulates data produced by other tasks and subsequently generates new data on which other tasks may depend on. With data in the picture, precedence constraints mean that a task cannot start execution before the tasks that it depends on are completed and the data it needs has arrived.

For each workflow, the precedence constraints are usually represented by a directed acyclic graph (DAG) (see Fig 1). A node of the DAG represents a task of the workflow. A directed edge from node i to node j means that task j depends on task i, and in particular, on the data that task i produces.

In typical workflow applications, the SLA is usually set at the workflow level rather than the task level. We assume each workflow has a deadline requirement; but a task does not have a deadline. A task has minimum computation resource requirements, such as the CPU power and memory.

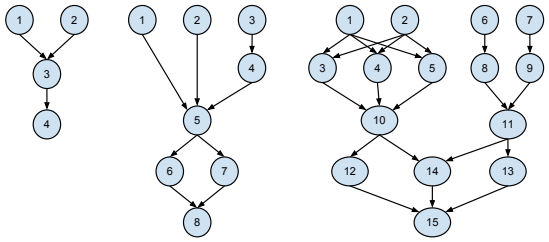

Figure 1. Three workflows

For ease of discussion, the computing engines in this paper are a set of VMs of different types. But, extension can be made to allow other types of computing engines, including physical machines and server clusters. The execution time of a task depends on the amount of computing resources of the VM on which the task is assigned to.

Our workflow scheduling problem is to decide when and on which VM to execute each task subject to important constraints: the precedence constraints including data availability, constraints about minimum resource requirements, SLA in the form of ready times and deadlines, and resource capacity constraints at the VMs. We will formulate the problem as an MIP optimization problem, so that among all the feasible solutions that satisfy all the constraints, we can find an optimal solution. The objective function will capture the operating cost or payment made to use the VMs.

We make two important assumptions:

- non-overlapping: at most one task runs on a VM at any time;
- non-preemption: a running task will not be interrupted.

With the two assumptions, once a task is scheduled to run on a VM starting at time t, it will continue to run until completion. Furthermore, it will be the only task running on the assigned VM during that time interval. We will later write constraints to enforce the non-overlapping assumption.

Their are reasons for imposing the non-overlapping constraint. First, the in-advance scheduling considered in this paper naturally applies to workflows where the individual tasks are "large", i.e., taking long time to run even on very powerful VMs. Once a VM is dedicated to such a task, there are no spare resources to run other tasks without significantly impacting the focal task. Running multiple tasks concurrently on a VM leads to unpredictable performance and makes it even more difficult to properly schedule the workflows. Second, with the interdependency of the tasks in a workflow, it is beneficial to complete a task as soon as possible so that other tasks that depend on it can be started sooner. The non-overlapping constraint helps to reduce the execution time of a task, when compared with running multiple tasks on the same VM at the same time.

With respect the second assumption, it is possible that improvement can be made by allowing preemption. For instance, if a non-preemptive schedule has periods of inactivity at some of the VMs, a preemptive schedule may allow some tasks to fill in those idle periods, but be preempted later. However, in practice, preemption leads to system overheads and it is difficult to manage. Preemption also makes the scheduling problem much more difficult. Nevertheless, it is still an interesting topic for future research.

B. Formulating the Workflow Scheduling Problem

Often-used notations are summarized in Table I.

Table I
OFTEN-USED NOTATIONS

$\mathcal{T}$	the set of time slots $\mathcal{T} = \{1, 2, \ldots, T\}$
$\mathcal{W}$	the set of workflows $\mathcal{W} = \{1, 2, \ldots, W\}$
$\mathcal{S}$	the set of all tasks of all workflows $\mathcal{S} = \{1, 2, \ldots, S\}$
$\mathcal{V}$	the set of VMs $\mathcal{V} = \{1, 2, \ldots, V\}$
A_w	the ready time of workflow w
D_w	the deadline of workflow w
C_k	the number of vCPUs of VM k
B_k	the amount of memory of VM k
c_j	the minimum number of vCPU required by task j
b_j	the minimum memory required by task j
P_k	the leasing price or other cost to operate VM k
R_{jk}	the running time of task j on VM k
l_{ij}	$l_{ij} = 1$ if task i depends on task j; $l_{ij} = 0$ otherwise
U_{ij}	data transfer time from task i to j if i and j are not co-located
$h_w(s)$	the value of workflow w if it is finished at time $A_w + s - 1$
Δ	duration of a time slot in minutes
L	the number of times slots per time frame (i.e., per hour)
$\mathcal{M}$	the set of time frames $\mathcal{M} = \{1, 2, \ldots, M\}$
x_{jk}^t	$x_{jk}^t = 1$ if and only if task j is scheduled to run on VM k starting at time slot t
y_k	$y_k = 1$ if and only if some tasks are assigned to VM k
z_{ij}	$z_{ij} = 0$ if and only if tasks i and j are assigned to the same VM, i.e., co-located
v_{ik}	$v_{ik} = 1$ if and only if task i is assigned to VM k
y_k^m	$y_k^m = 1$ if and only if VM k is used on time frame m

1) Time Structure: We will consider a discrete-time model, where time is divided into a sequence of time slots $1, 2, \ldots, T$ of identical duration, e.g., 5 minutes per time slot. Here, T corresponds to the furtherest time horizon for the scheduling problem. For instance, if we schedule workflows for the next 24 hours and if the time slot size is 5 minutes, then $T = 12 \times 24 = 288$. The choice of a discrete-time model is in part due to the convenience it offers in formulating the problem. This will become clear later. Let $\mathcal{T} = \{1, 2, \ldots, T\}$.

2) Main Input: Consider a set of cloud computing workflows $\mathcal{W} = \{1, 2, \ldots, W\}$. Each workflow contains one or more tasks, and it can be viewed as a set of tasks. The total pool of tasks from all the workflows is denoted by the set $\mathcal{S} = \{1, 2, \ldots, S\}$. Each task $j \in \mathcal{S}$ can only belong to one workflow.

We represent the task precedence constraints by a 0-1 matrix $(l_{ij}),\ \forall i, j \in \mathcal{S}$, where $l_{ij} = 1$ if task i depends on task j and $l_{ij} = 0$ if task i does not depend on task j. By convention, we set $l_{ii} = 0$ for each i. The meaning of the precedence constraints is as follows: If $l_{ij} = 1$, then the start time of task i should be after the finish time of task j plus the data transfer time from j to i.

Each workflow w has a deadline denoted by D_w. It also has a ready time, denoted by A_w.

For computing resources in the cloud, we assume there is a set of virtual machines (VMs) of different types and capabilities. The set is denoted by $\mathcal{V} = \{1, 2, \ldots, V\}$. For each VM $k \in \mathcal{V}$, let C_k be the number of vCPUs it supports and let B_k be the amount of its memory.

3) Main Decision Variables: The workflow scheduling problem is to assign tasks to the VMs at the right time so that the precedence constraints, resource constraints and timing requirements are all satisfied.

Because of the non-overlapping and non-preemption assumptions, once a task is scheduled to run on a VM k starting at time t, it will continue to run on that VM and use the VM exclusively until completion. Thus, to schedule the workflows, it is enough to decide the start times and the VM assignments of all the tasks.

With the help of the discrete-time structure, we use the binary decision variables x_{jk}^t to represent the task assignment: $x_{jk}^t = 1$ if task j is scheduled to run on VM k starting at time slot t, and $x_{jk}^t = 0$ otherwise.

Each task should start only once, which can be specified as:

$$\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} x_{jk}^t = 1,\ \forall j \in \mathcal{S}. \tag{1}$$

Consider a fixed task j. Since the variables x_{jk}^t can take values of only 0 or 1, (1) is the same as saying that exactly one x_{jk}^t will take the value 1; that is, task j will start exactly once on exactly one VM.

Let the binary variable y_k denote the on/off (active/idle) status of each VM k, with $y_k = 1$ if some tasks are assigned to VM k at any time, and $y_k = 0$ otherwise. If VM k has no task assignment, then we can potentially suspend the VM or terminate the VM's lease to reduce cost. However, if one or

more task is scheduled to run on the VM, it has to be turned on ($y_k = 1$).

We need the following constraints.

$$\sum_{j \in \mathcal{S}} \sum_{t \in \mathcal{T}} x_{jk}^t \leq y_k |\mathcal{S}|, \ \forall k \in \mathcal{V}. \tag{2}$$

Clearly, $y_k = 0$ forces $x_{jk}^t = 0$ for all $j \in \mathcal{S}$ and all $t \in \mathcal{T}$. When $y_k = 1$, the constraint in (2) allows the possibility that $x_{jk}^t = 0$ for all $j \in \mathcal{S}$ and all $t \in \mathcal{T}$; that is, while VM k is turned on, no tasks are ever assigned to it. However, when combined with the optimization objective of minimizing the cost of running the active VMs, such a solution is not optimal. Thus, any optimal solution in fact will satisfy (2) as equalities; consequently, if $y_k = 1$, then $x_{jk}^t = 1$ for some j and some t.

4) Resource Constraints: Each task requires certain minimum amount of resources to run properly. For example, a task may require a minimum of 2 vCPUs and 4 GB memory to run. For each task $j \in \mathcal{S}$, let c_j be the minimum number of vCPU required, and let b_j be the minimum memory requirement.

Each VM k has certain resource capacity in terms of the number of vCPUs C_k and memory capacity B_k in GB (and local disk/SSD volume in GB, which we omit for brevity). From the perspective of each task, the resource constraints state that the assigned VM for the task must have sufficient resources to run the task:

$$\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} C_k x_{jk}^t \geq c_j, \ \forall j \in \mathcal{S} \tag{3}$$

$$\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} B_k x_{jk}^t \geq b_j, \ \forall j \in \mathcal{S}. \tag{4}$$

Inequality (3) and (4) are for the CPU and memory, respectively. Since, for each j, exactly one x_{jk}^t takes the value 1, (3) really says that $C_k \geq c_j$ for k such that $x_{jk}^t = 1$ for some t. That is, if task j is assigned to VM k, then $C_k \geq c_j$. A similar implication can be said about (4).

Combining with the non-overlapping assumption, (3) and (4) also represent the capacity constraints for each VM k, i.e., the capacity of each resource on VM k is not exceeded.

5) Non-Overlapping Constraints: The non-overlapping constraints mean that, for each time slot t and each VM k, at most one task occupies time slot t on VM k.

For each $j \in \mathcal{S}$ and $k \in \mathcal{V}$, let R_{jk} denote the running time of task j on VM k, which is assumed to be given. If R_{jk} is inversely proportional to C_k, we have $R_{jk} = \lambda_j / C_k$, where λ_j is the amount of computation required by task j measured in vCPU-hours. For instance, if it takes 4 hours to run the task on a VM with 3 vCPUs, then $\lambda_j = 12$ vCPU-hours. If the same task is executed on a VM with 6 vCPUs, then $R_{jk} = 2$ hours. In more general situations, R_{jk} may not be inversely proportional to C_k. The data for R_{jk} can be supplied in advance by measuring or estimating the running times of each task j on different VM types based on either trials or historical data.

With that, the non-overlapping constraints can be specified:

$$\sum_{j \in \mathcal{S}} \sum_{r=\max(1, t-R_{jk}+1)}^{t} x_{jk}^r \leq 1, \ \forall k \in \mathcal{V}, \forall t \in \mathcal{T}. \tag{5}$$

We will explain why (5) is the right specification for non-overlapping under the assumption of non-preemption. The execution of task j occupies time t on VM k if and only if it is assigned to VM k with the start time in the set $\{\max(1, t - R_{jk} + 1), \ldots, t\}$. The latter holds if and only if $\sum_{r=\max(1,t-R_{jk}+1)}^{t} x_{jk}^r = 1$. Thus, (5) means that there is at most one task occupying time t on VM k.

6) Precedence Constraints: For each pair of tasks i and j where j depends on i (i.e., with $l_{ji} = 1$), let U_{ij} denote the data transfer time from task i to task j, provided tasks i and j are not assigned to the same VM. We consider a scenario where all the VMs are in the same datacenter. With that, it is reasonable to assume that the bandwidth between the VMs are roughly equal, and therefore, the data transfer time U_{ij} is proportional to the amount of data that task j needs from task i. Since that amount of data can be known, or estimated, ahead of time, each U_{ij} is assumed to be given.

The precedence constraints can be formulated as follows:

$$\big(\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} t \, x_{ik}^t - \sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} (t + R_{jk}) x_{jk}^t - U_{ji} z_{ij} \big) l_{ij} \geq 0, \quad \forall i, j \in \mathcal{S}. \tag{6}$$

Here, when task i does not depend on task j, where $l_{ij} = 0$, the inequality is vacuous. When task i depends on j, it says task i must start after the completion time of task j plus the data transfer time from j to i. Note that $\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} t \, x_{ik}^t$ is the start time of task i; $\sum_{k \in \mathcal{V}} \sum_{t \in \mathcal{T}} (t + R_{jk}) x_{jk}^t$ is the completion time of task j plus one.

Each z_{ij} is a 0-1 variable indicating whether tasks i and j will be co-located. Specifically, $z_{ij} = 0$ means that both tasks i and j are assigned to the same VM, and $z_{ij} = 1$ means otherwise. Suppose task i depends on j ($l_{ij} = 1$). When task i and j are not co-located, $U_{ji} z_{ij} = U_{ji}$; when they are co-located, $U_{ij} z_{ij} = 0$, in which case there is no need to transfer the data from task j to task i. Thus, $U_{ji} z_{ij}$ is the data transfer time from task j to task i regardless of whether the two tasks are co-located.

We next describe the conditions that the z_{ij} variables must satisfy in order to have the intended physical meaning (about co-location). For ease of presentation, we introduce another set of binary variables v_{ik}, defined by

$$v_{ik} = \sum_{t \in \mathcal{T}} x_{ik}^t, \quad \forall i \in \mathcal{S}, \forall k \in \mathcal{V}. \tag{7}$$

Note that each v_{ik} indicates whether task i is assigned to VM k, with $v_{ik} = 1$ if and only if task i is assigned to VM k.

We will write the constraints that ensure $z_{ij} = 0$ if and only if task i and task j are assigned to the same VM. We first require

$$-|\mathcal{V}|z_{ij} \leq \sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \leq |\mathcal{V}|z_{ij},\ \forall i,j \in \mathcal{S}. \quad (8)$$

When $z_{ij} = 0$, (8) implies $\sum_{k\in\mathcal{V}} kv_{ik} = \sum_{k\in\mathcal{V}} kv_{jk}$, which implies $v_{ik} = v_{jk}$ for some k. When $z_{ij} = 1$, (8) poses no constraints: It is always satisfied for our case where the sets of binary variables $\{v_{ik}\}_k$ and $\{v_{jk}\}_k$ each have exactly one 1. It is easy to see that, for such $\{v_{ik}\}_k$ and $\{v_{jk}\}_k$,

$$1 - |\mathcal{V}| \leq \sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \leq |\mathcal{V}| - 1.$$

The next set of constraints ensures that when $z_{ij} = 1$, task i and task j are not assigned to the same VM:

$$|\sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk}| \geq z_{ij},\ \forall i,j \in \mathcal{S}. \quad (9)$$

In order to use standard MIP solvers, we need to convert (9) to equivalent linear constraints. First, for each pair of i and j, (9) is equivalent to the following two *disjunctive* constraints:

$$\sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \geq z_{ij} \quad (10)$$

$$\text{or } \sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \leq -z_{ij}. \quad (11)$$

To combine such disjunctive inequalities into normal (conjunctive) inequalities, for each pair of i and j, let θ_{ij} be a 0-1 variable, taking the value 1 if (10) is active and the value 0 if (11) is active. We then have, for each pair of i and j,

$$\sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \geq z_{ij} - |\mathcal{V}|(1-\theta_{ij}) \quad (12)$$

$$\sum_{k\in\mathcal{V}} kv_{ik} - \sum_{k\in\mathcal{V}} kv_{jk} \leq -z_{ij} + |\mathcal{V}|\theta_{ij}. \quad (13)$$

When $\theta_{ij} = 1$, (12) is the same as (10) and (13) always holds. When $\theta_{ij} = 0$, (13) is the same as (11) and (12) always holds.

7) Ready Time Constraint: Each workflow has a ready time, which is the earliest time when it is ready to be executed. This normally means that the prerequisites of the workflow are satisfied. For instance, for a workflow that conducts hourly or daily data analysis, the workflow depends on the data from the previous hour or day, and it is only ready after the required data is available in the system.

The ready time of a workflow w is denoted by A_w. We require:

$$\sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}} t\, x^t_{jk} \geq A_w\,, \forall w \in \mathcal{W}, \forall j \in w. \quad (14)$$

That is, no task of a workflow w can be scheduled to start earlier than A_w.

8) Deadline Constraints: Deadline constraints are very important in workflow scheduling, since the deadline is one of the most important aspects of SLA. Violating a workflow deadline may reduce the usefulness of the workflow results, incur late penalty, cause loss of profit, and in some situations even render the entire workflow worthless.

One of the commonly used notions of deadline is a hard deadline, where a workflow is required to be finished by its deadline. Hard deadlines apply to situations where a late workflow is not only worthless but may incur steep penalty. In the case where there is no possible way to satisfy all workflow deadlines, the scheduler needs to generate warning in advance so that some workflows are rejected when they enter the system, a process known as admission control.

A workflow w is completed by its deadline D_w if and only if every task j of workflow w is completed by D_w. Thus, the deadline constraints can be written as

$$\sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}} (t + R_{jk} - 1)x^t_{jk} \leq D_w,\ \forall w \in \mathcal{W}, \forall j \in w. \quad (15)$$

C. Putting It Together: the Min-Cost Problem

Let P_k be the leasing price or other cost to operate VM k (such as due to power consumption). Our objective here is to minimize the total price or operating cost. The complete problem formulation is as follows:

$$\min \sum_{k\in\mathcal{V}} P_k\, y_k \quad \textbf{(Min-Cost)}$$

$$\text{s.t. } (1)(2)(3)(4)(5)(6)(7)(8)(12)(13)(14)(15)$$

$$x^t_{jk}, y_k, z_{ij}, v_{ik}, \theta_{ij} \in \{0,1\},\ \forall i,j \in \mathcal{S}, \forall k \in \mathcal{V}, \forall t \in \mathcal{T}.$$

III. More Advanced Formulations

A. Soft Deadline and Profit Maximization

Instead of hard deadline constraints, there are alternatively models that impose "soft" deadline constraints. More generally, we will consider the following profit-maximization problem, where the objective is to maximize the difference between the total value gained from completing the workflows and the total cost.

Suppose that the completion of a workflow generates a value that depends on its completion time relative to its ready time. Specially, consider the workflow w with ready time A_w. Let $h_w : \{1, 2, \ldots\} \to \mathbb{R}$ be a non-increasing *value function*, where $h_w(s)$ represents the value gained when workflow w is finished at time $A_w + s - 1$. We assume $h_w(1) > 0$ for each w.

In this setup, there may still be a deadline D_w. But, it is understood as a soft deadline. For instance, we can set $h_w(s) = m_1 > 0$ for $1 \leq s \leq D_w - A_w + 1$ and $h_w(s) \leq m_2$ for $s > D_w - A_w + 1$, where m_1 and m_2 are constants with $m_1 > m_2$. Then, there is a constant gain when workflow w is finished by the deadline D_w. Depending on the values of h_w for $s > D_w - A_w + 1$, there is either a reduced gain,

no gain, or a penalty if the workflow is finished after the deadline.

The value-function-based approach is far more flexible and useful than the hard-deadline problems. It allows the system to execute late workflows at a reduced gain or with a penalty. The gain or penalty may depend on the tardiness of the workflow. Moreover, a hard-deadline problem becomes a special case if, for each workflow w, we set $h_w(s)$ to some negative value with sufficiently large magnitude for $s > D_w - A_w + 1$.

The hard and soft deadlines can be easily combined into a single problem, where some workflows enjoy deadline guarantee and others take the value-function-based approach.

We next give the formulation. For each workflow $w \in \mathcal{W}$ and each time $t \in \mathcal{T}$, define a binary variable u_w^t. The intention is to have $u_w^t = 1$ if and only if workflow w is completed at time t. Then, the completion time of the workflow can be written as $\sum_{t\in\mathcal{T}} t u_w^t$.

The profit-maximization problem is as follows:

$$\max \sum_{w\in\mathcal{W}}\sum_{t\in\mathcal{T}} h_w(t - A_w + 1)u_w^t - \sum_{k\in\mathcal{V}} P_k y_k \quad \textbf{(Max-Profit)}$$

$$\text{s.t.} \sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}} (t + R_{jk} - 1)x_{jk}^t \le \sum_{t\in\mathcal{T}} t u_w^t, \quad \forall w \in \mathcal{W}, \forall j \in w \qquad (16)$$

$$(1)(2)(3)(4)(5)(6)(7)(8)(12)(13)(14).$$

The decision variables are now $x_{jk}^t, y_k, z_{ij}, v_{ik}, \theta_{ij}$ and u_w^t. Compared with the min-cost problem, the constraints (16) are added and the deadline constraints (15) are removed.

Remark. Recall that $\sum_{k\in\mathcal{V}}\sum_{t\in\mathcal{T}}(t + R_{jk} - 1)x_{jk}^t$ is the completion time of task j. Expression (16) says that the workflow completion time must be an upper bound for the task completion times for all the tasks of the workflow. For $\sum_{t\in\mathcal{T}} t u_w^t$ to be the completion time of workflow w, at least one of the inequalities (for some $j \in w$) in (16) must be binding, i.e., becomes equality. However, in the above formulation, there are no expressions enforcing the binding requirement. If h_w is strictly decreasing for every w, then any optimal solution must satisfy the binding condition; for otherwise, the objective value can be increased by setting $u_w^t = 1$ for some earlier time t and some w, which leads to a reduction of $\sum_{t\in\mathcal{T}} t u_w^t$. If h_w is non-increasing for some w, it is possible that, in an optimal solution, all the inequalities in (16) are strict for some w; that is, $\sum_{t\in\mathcal{T}} t u_w^t > t_w$, where t_w denotes the true workflow completion time. In that case, it must be that h_w has identical values for $t = t_w - A_w + 1, t_w - A_w + 2, \ldots, \sum_{t\in\mathcal{T}} t u_w^t - A_w + 1$.

B. On-Demand VM Payment Model

Amazon AWS has two VM leasing models with different pricing schemes. One is long-term lease with a lower per-hour cost, which is charged regardless of whether the VM is used or not. The other is the on-demand or pay-as-you-go model, where a customer can turn on or off VMs as his workload varies in order to save money. However, the per-hour cost is higher with the on-demand model.

The formulations in earlier sections are suitable for the long-term lease model. We next present a formulation that captures the on-demand payment model. For the on-demand model, the main issue is that the time slot size is usually much shorter than one hour, e.g. 5 minutes or 1 minute. As the payment granularity is an hour, a VM will incur one-hour cost even if it is actually used for only one time slot. An optimal schedule will minimize such undesirable usage of the VMs.

Suppose the duration of a time slot is Δ minutes and suppose Δ is a factor of 60. Then, each hour contains $L \triangleq 60/\Delta$ time slots. We organize the time slots into $M \triangleq \lceil T/L \rceil$ time frames, with each frame consisting of L times lots. The set of time frames is denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$. For ease of presentation, we assume T is divisible by L so that $M = T/L$.

We extend the y_k variables to the y_k^m variables, with $y_k^m = 1$ if and only if VM k is used/active on time frame m. We will specify constraints that replace those in (2).

For y_k^m to have the intended meaning, we require

$$\sum_{t=(m-1)L+1}^{mL}\sum_{j\in\mathcal{S}}\sum_{r=\max(1,t-R_{jk}+1)}^{t} x_{jk}^r \le L\, y_k^m, \quad \forall k \in \mathcal{V}, \forall m \in \mathcal{M}. \qquad (17)$$

To see this is correct, suppose $y_k^m = 0$, which means VM k is not used on the time slots $t = (m-1)L+1, \ldots, mL$. That is, for each such t, each task j cannot be running on VM k at time t, which is the case if and only if the start time of task j does not fall in the set $\{\max(1, t-R_{jk}+1), \ldots, t\}$. From (17), $y_k^m = 0$ implies $\sum_{r=\max(1,t-R_{jk}+1)}^{t} x_{jk}^r = 0$ for each $j \in \mathcal{S}$ and each $t \in \{(m-1)L+1, \ldots, mL\}$, which further implies $x_{jk}^r = 0$ each $r \in \{\max(1, t - R_{jk} + 1), \ldots, t\}$ for the given j and t.

Next, consider the case of $y_k^m = 1$. By the non-overlapping constraints in (5), for each $k \in \mathcal{V}$ and each $t \in \{(m-1)L + 1, \ldots, mL\}$,

$$\sum_{j\in\mathcal{S}}\sum_{r=\max(1,t-R_{jk}+1)}^{t} x_{jk}^r \le 1.$$

Thus, for each $k \in \mathcal{V}$,

$$\sum_{t=(m-1)L+1}^{mL}\sum_{j\in\mathcal{S}}\sum_{r=\max(1,t-R_{jk}+1)}^{t} x_{jk}^r \le L.$$

Therefore, in the case of $y_k^m = 1$, (17) poses no additional constraints.

1) Min-Cost Formulation: Under the on-demand payment model, the min-cost problem can be formulated as follows.

$$\min \sum_{k\in\mathcal{V}}\sum_{m\in\mathcal{M}} P_k\, y_k^m$$

$$\text{s.t. } (1)(2)(3)(4)(5)(6)(7)(8)(12)(13)(14)(15)(17).$$

Table II
AMAZON EC2 VM TYPES

VM Type	vCPU	Memory (GB)	On-Demand $	Reserve $
m4.large	2	8	0.126	0.085
m4.xlarge	4	16	0.251	0.172
m4.2xlarge	8	32	0.503	0.342
m4.4xlarge	16	64	1.005	0.684
m4.10xlarge	40	160	2.514	1.709
m4.16xlarge	64	156	4.022	2.735

The decision variables are now $x^t_{jk}, y^m_k, z_{ij}, v_{ik}$, and θ_{ij}.

The max-profit problem under the on-demand payment model can be similarly formulated by incorporating (17).

The Amazon AWS long-term lease requires monthly or yearly contract. To compare the two payment models, we would have to know the long-term workflow pattern. Alternatively, we can assume the same set of workflows repeats daily. The latter is particularly relevant for daily big data analytics.

IV. NUMERICAL EVALUATIONS AND COMPARISON

In this section, we report numerical experiments on the workflow problems. The goal is to evaluate the effectiveness of the MIP approach for solving them.

The VM types that we used (see Table II) are a subset of the VM types offered by Amazon EC2. The on-demand pricing and one-year reserved pricing are listed in the last two columns in US dollars per hour. For each of our experiments, we provide a different set of VMs of the types in Table II for the scheduling algorithm to choose from. All the experiments were run on a PC with an AMD 3.5 GHz quad-core CPU and 8GB of memory. Gurobi 7.0 was used as the MIP solver.

In Section IV-A, we compare the performance of our MIP approach with a notable, related workflow scheduling algorithm. Section IV-B compares the results obtained for the two different payment models. In Section IV-C, we evaluate the runtime of the MIP algorithm and discuss possible improvement.

A. Comparison with Heuristic Algorithm

Here, we compare the performance of the MIP approach against the IC-PCP algorithm reported in [6]. IC-PCP is a critical-path-based heuristic algorithm for scheduling workflows with deadline constraints. Its objective is to minimize cost under the on-demand payment model. Thus, it is appropriate to compare IC-PCP with MIP for the min-cost problem that we formulated. Note that the IC-PCP algorithm is less capable than the MIP-based approach. It works for a single workflow, and it does not consider the data transfer time between dependent tasks. To make a fair comparison, we used the scientific workflows described in [6], [17] and ignored data transfer between tasks. Each experiment contains only one workflow of 'LIGO', 'Montage', 'SIPHT', or 'Epigenomics'

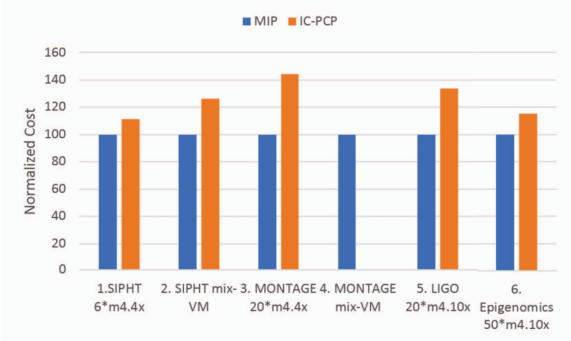

Figure 2. Performance comparison on the min-cost problem

We set the time slot size to be 5 minutes and a charging period (an hour) contains 12 time slots. We round fractional numbers up so that each task takes an integer number of time slots to finish.

Fig. 2 shows the comparison results of the two approaches. The total costs are normalized with the total cost from the MIP algorithm being 100.

The first two experiments in Fig. 2 are both for the SIPHT workflow, with different sets of VMs and deadlines. The first one is given 6 m4.4xlarge VMs; the second one is given a mix of 4 m4.xlarge, 4 m4.2xlarge and 4 m4.4xlarge VMs. The MIP algorithm gives optimal solutions in all circumstances, provided the problem is feasible. But, the critical-path-based algorithms such as IC-PCP only perform well when the VMs are the same or similar in capabilities. The main reason is that scheduling and path selection in those algorithms are largely based on estimating the task completion times. When the VMs are heterogeneous, i.e., having very different capabilities, the execution times of the tasks depend on the assigned VM types, and that dependency is not treated in IC-PCP. This leads to the use of highly inaccurate estimates in the algorithm, which in turn leads to sub-optimal selection of paths and poor assignment of tasks, resulting in much larger costs. In experiments 3 and 4, which use the MONTAGE workflow and have heterogeneous VMs and tight deadlines, IC-PCP failed to find feasible schedules, while the MIP algorithm not only found feasible schedules but also optimal ones. In experiments 5 and 6, all the VMs are of the same type for each experiment.

We have the following general observations from the results.

- The MIP algorithm consistently finds optimal solutions whenever the problem is feasible.
- Critical-path-based algorithms rely heavily on accurate estimation of task/path completion times in order to make reasonable scheduling choice. When the VMs are heterogeneous in capabilities, it is difficult to accurately estimate the execution times without knowing the actual VM assignments; thus the scheduling decision may be far from being optimal.

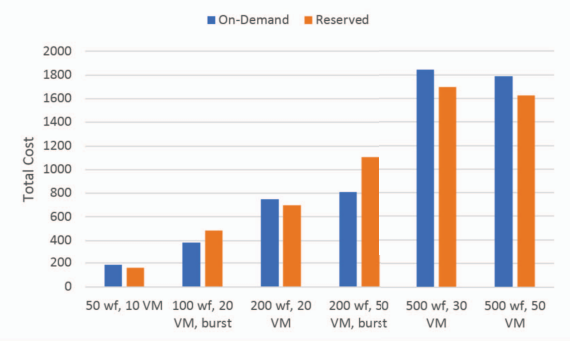

Figure 3. Performance with different payment models

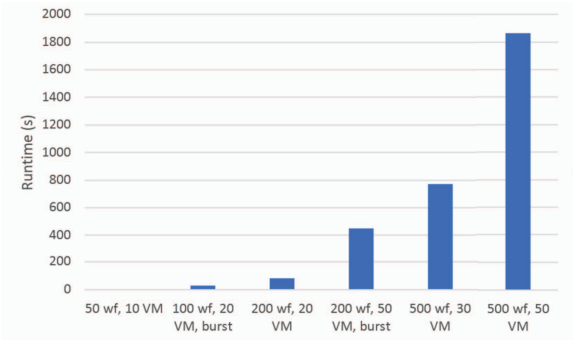

Figure 4. Runtime of MIP algorithm

- Even when using the same type of VMs, the heuristic algorithm does not give consistent performance. For example, IC-PCP returns 11% to 43% higher costs across the experiments. Since the workflow problem needs to satisfy multiple difficult constraints, simple heuristics are unable to take care of all these constraints while optimizing the objective function.

B. Comparison of Different Payment Models

Out of the two payment models in Section IV-B, customers can easily compare their costs by experimenting with the formulations we provided and determine which payment model is more suitable for their workflows and use cases.

In this set of experiments, we use three types of workflows shown in Fig. 1. They each contain 4, 8 and 15 tasks, respectively. Here, we have moved away from the scientific workflows described in [17], but towards more contemporary big-data analytic workflows, such as MapReduce workflows. In such a setting, each workflow is more likely to have a few to a few dozens large tasks and each task often consists of many smaller, independent sub-tasks that can run in parallel.

In each of the experiments, we use the MIP algorithm on a mixed set of workflows from Fig. 1 and generate results for both the on-demand payment model and the reserved (long-term lease) model. Each workflow has its own ready time and deadline. The data transfer times between dependent tasks are also taken into consideration. We set the time horizon to 24 hours to better imitate the daily workflow activities in industry.

Among the six experiments, the second and fourth use input with bursty workflow requests. That is, a large number of workflow requests take place within a short period of time and the rest of time in the 24 hours is relatively quiet. As we can see from the results on the total cost in Fig. 3, the on-demand model suits this use case well. It yields a lower cost for the customer, since more VMs can be requested during the high-activity hours and idle VMs are turned off during the low-activity hours. The other four experiments each have evenly distributed workflow requests. As a result, the long-term lease model leads to a lower cost and relative high utilization through the entire 24-hour time. Such direct comparison can help customers make better choices of the payment model based on their workflow request patterns.

C. Runtime and Scalability of MIP Algorithm

Any non-trivial class of workflow scheduling problems is almost certainly NP-hard, which implies that any optimal algorithm, including the MIP algorithms, will likely be time-consuming as the problem size becomes large. To evaluate the scalability of the MIP approach on the formulated problems, we experimented with different problem sizes. The runtime results are shown in Fig. 4.

As we can see from the figure, the algorithm runtime is fairly short when the problem size is small. The first three experiments each finished within a few seconds to a minute. However, when the problem size increases, the runtime increases rapidly. The last experiment has 500 workflows and 50 available VMs. It took more than 1865 seconds to finish.

The long runtime for large problems is still acceptable when the workflow pattern is relatively static each day, as in many industrial use cases. The computation only needs to be done once, or done infrequently. Overall, with the way we set up the parameters (such as the time slot size), the MIP approach is suitable when there are a few hundred daily workflows, each with dozens of tasks, and the number of VMs are dozens to hundreds.

To reduce the algorithm runtime and widen the applicability of MIP, there are several possibilities for improvement.

- Combine similar or parallel tasks into one task within a workflow. This leads to fewer tasks in the workflow and hence fewer variables in the MIP problem.
- Reduce the time horizon or increase the time slot size. Both can reduce the number of time slots, thus, the number of variables.
- Use non-uniform time slot sizes, while keeping the time horizon unchanged. For instance, one can use finer time slot sizes during busy business hours, and coarser time slot sizes during quieter time.

- Decompose a large problem into multiple smaller problems, each of which schedules a subset of the workflows on a subset of the VMs. The smaller problems can be solved in parallel on multiple servers. Although such decomposition introduces sub-optimality, each of the smaller scheduling problems is still solved optimally.

V. Conclusions

The goal of this paper is to explore the suitability of using MIP formulations and algorithms to schedule complex workflows in the cloud. As explained throughout the paper, such workflow scheduling problems can be very challenging because they involve many difficult constraints. It is not easy to develop good heuristic algorithms for solving such problems. Known heuristic algorithms generally work on simpler versions of the problems, and even in those cases, often fail to find feasible solutions and/or are under-achieving in terms of the optimization objective.

From this study, we can conclude that for problems up to certain size, the MIP approach is entirely applicable. With MIP, one can describe the problem precisely and then resort to MIP algorithms to find not only feasible but optimal solutions. For use cases where the problems are solvable by MIP, there is little need to look for heuristic algorithms. It is only when the use cases generate large problems that heuristic algorithms become useful. As explained at the end of Section IV-C, one can also use MIP-based heuristics, such as decomposition, to solve large problems.

The workflow scheduling problems considered here are of the in-advance reservation type. To cope with workflow dynamics or to deal with a large number of small workflows or small tasks, real-time dynamic scheduling algorithms are still needed. All big-data analytic platforms such as Hadoop have the latter type of schedulers, which are usually based on simple heuristic algorithms.

Acknowledgment

The research is supported in part by the National Science Foundation of US under grant STC-1562485.

References

[1] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on Amazon EC2," *Journal of Grid Computing*, vol. 10, no. 1, pp. 5–21, Mar. 2012.

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

[3] Hadoop, http://hadoop.apache.org/.

[4] Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 306–319, July 2014.

[5] L. Zeng, B. Veeravalli, and X. Li, "Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud," in *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, March 2012, pp. 534–541.

[6] 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.

[7] J. Sahni and D. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a Cloud environment," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, 2015.

[8] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[9] T. Sandholm and K. Lai, "Dynamic proportional share scheduling in Hadoop," in *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP'10, 2010.

[10] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[11] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011.

[12] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, 2012.

[13] P. Hoenisch, C. Hochreiner, D. Schuller, S. Schulte, J. Mendling, and S. Dustdar, "Cost-efficient scheduling of elastic processes in hybrid clouds," in *IEEE International Conference on Cloud Computing (CLOUD)*, June 2015.

[14] A. Ruiz-Alvarez, I. K. Kim, and M. Humphrey, "Toward optimal resource provisioning for cloud MapReduce and hybrid cloud applications," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 669–677.

[15] L. Thai, B. Varghese, and A. Barker, "Budget constrained execution of multiple bag-of-tasks applications on the cloud," in *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 975–980.

[16] J. Jiang, S. Ma, B. Li, and B. Li, "Symbiosis: Network-aware task scheduling in data-parallel frameworks," in *IEEE INFOCOM*, April 2016, pp. 1–9.

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