On Optimal Scheduling Algorithms for Well-Structured Workflows in the Cloud with Budget and Deadline Constraints

Yang Wang

Center for Cloud Computing
Shenzhen Institute of Advanced Technology
Chinese Academy of Science, China.

E-mail: yang.wang1@siat.ac.cn

Wei Shi

Faculty of Business and I.T.
University of Ontario Institute of Technology
Oshawa, ON, Canada

E-mail: wei.shi@uoit.ca

Kenneth B. Kent

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada

E-mail: ken@unb.ca

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

In this paper, we consider optimal scheduling algorithms for scientific workflows with two typical structures, fork&join and tree, on a set of provisioned (virtual) machines under budget and deadline constraints in cloud computing. First, given a total budget B, by leveraging a bi-step dynamic programming technique, we propose optimal algorithms in pseudo-polynomial time for both workflows with minimum scheduling length as a goal. Our algorithms are efficient if the total budget B is polynomially bounded by the number of jobs in respective workflows, which is usually the common case in practice. Second, we consider the dual of this optimization problem to minimize the cost when the deadline of the computation D is fixed. We change this problem into the standard multiple-choice knapsack problem via a parallel transformation.

Keywords: workflow scheduling; budget and deadline constraints; optimal scheduling algorithm; dynamic programming; cloud

1. Introduction

The cloud, as a new computing platform with abundant on-demand compute resources and elastic charging models, has been emerging as a promising approach

to the high performance workflow computation [10, 12, 13, 8]. Although this new platform removes the up-front fee and long-term commitment of installing and maintaining hardware and software infrastructure while providing HPC applications with a scalable runtime environment, it still imposes some challenges that have not seen before. One of these challenges is cost-effective utilization of the cloud resources.

With the advances of more commercial cloud systems penetrating into the scientific computation market, research on this aspect has been provoking great interest [26, 5, 28]. Yu et al. [26] discussed this problem in the context of Service Grids where a QoS-based workflow scheduling method is present to minimize the execution cost and yet meet the time constraints imposed by the user. In contrast, Zeng et al. [28] considered the executions of large scale many-task workflows in the cloud with budget constraints. To effectively balances the execution time-and-monetary costs, they proposed *ScaleStar*, a budget-conscious scheduling algorithm, which assigns the selected task to a virtual machine with a higher comparative advantage. Almost at the same time, Caron et al. [5] studied the same problem for non-deterministic workflows. They presented a way of transforming the initial problem into a set of addressed sub-problems thereby proposing two new allocation algorithms for resource allocations under budget constraints. All of these studies focus on the scheduling of scientific workflows with a deterministic or non-deterministic DAG (directed acyclic graph) shape. In general, the optimization of this problem either with QoS- or budget-based constraints is NP-complete. Therefore, a variety of heuristics have been proposed for sub-optimal solutions, such as those aforementioned efforts. However, the existing research with interest in the efficient solutions to the optimal scheduling of *certain* types of workflows under QoS- or budget constraints is still few and far between. Some people may attribute this phenomenon to the complex of scientific workflows in terms of the data- or control-dependencies. However, in reality, this explanation is not always convincing. Some well-structured workflows are commonplace, and efficient optimal scheduling of these workflows exists. With optimal solutions, we can obtain insights into many facets of this optimization problem and by which to evaluate other heuristics.

In this paper, we consider the scheduling of two well-structured, yet often-used workflows, *fork&join* and *tree*, in the cloud with budget and deadline constraints. The fork&join workflow is a very common form of parallel scientific computing processes and represents a large class of problems with a pipeline of parallel phases [23, 2, 1]. It is typically characterized by multiple synchronized stages, namely, one stage cannot start until its immediately preceding stage is finished. Each stage consists of a collection of sequential or parallel jobs. A typical fork&join workflow in practice is those iterative MapReduce jobs that implement graph-based algorithms [16, 4, 9, 29, 14].

The tree structure is another commonplace workflow pattern, which is usually characterized by the average fan-out factor of each job in the workflow. As a hierarchical workflow model, tree structure is often used to capture several hierarchy

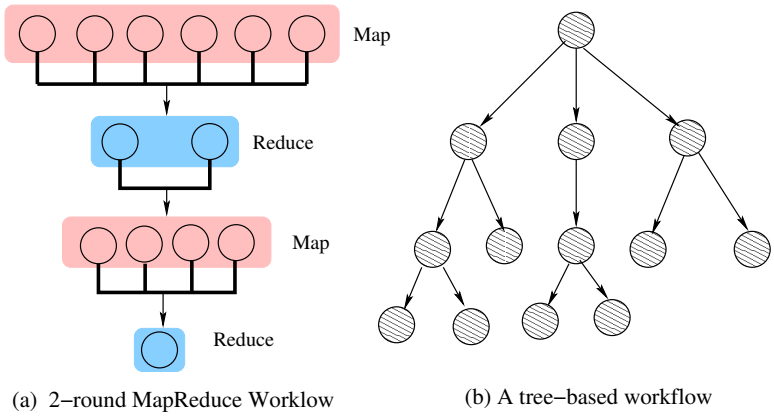


Fig. 1. Examples of studied workflows: fork&join and tree.

levels of a workflow in one model [3], and thus it is often found in recursive computation [21]. A real-live example is the elemination-tree workflow that is developed to represent the storage requirements, and computational dependencies in the Cholesky and LU factorization of sparse matrices [20, 11].

In cloud platforms the constituent jobs in the workflow can be scheduled on different machines for parallel executions. Each types of machine, depending on the performance or the configuration, has different service rates. Fig. 1 shows two examples of the studied workflows. One is a two-round MapReduce job which is represented by a 4-stage fork&join workflow, each having $6, 2, 4$, and 1 jobs (tasks in MapReduce jargon), and the other is a tree-based workflow, consisting of 13 jobs.

Since the resources of cloud computing are on-demand provisioned according to a typical "pay-as-you-go" model, the cost-effective selection and utilization of the resources used to compute their workloads are thus the major concerns of cloud users. In this paper, we try to address this problem by proposing optimal scheduling algorithms for such workflows on a set of (virtual) machines in the cloud, each with different performance and service rates. More specifically, we consider the following two optimization problems:

(1) Given a fixed budget B, how to efficiently select the machine from a candidate set for each job so that the total scheduling length of the workflow is minimal within the allotted budget;
(2) Given a fixed deadline D, how to efficiently select the machine from a candidate set for each job so that the total cost of the workflow is minimal within the allotted time;

At first glance, both problems are primary-dual to each other, and thus they have highly inter-dependent solutions, solving one is sufficient to solve the other. However, we will show that there are still some asymmetries in their solutions.

In this paper, we focus squarely on the first problem, and briefly discuss the sec-

ond one. To address the first problem, we develop a bi-step dynamic programming algorithm for the fork&join workflow where a stage-wise optimal solution to compute the minimum execution time with given budget for each stage can be obtained in parallel at the first step, and then a global optimal solution can be achieved at the second step. We then take advantages of these results and design the optimal algorithm for the tree workflow. As for the second problem, it is relatively straightforward for both workflows as for the fork&join, we can change it into the standard *multiple-choice knapsack* (MCKS) problem [22, 19] via a parallel transformation while for the tree, we can simply give a recursive algorithm based on the problem formulation.

The remainder of this paper is structured as follows. In the next section, we overview some related work to compare ours with others, and then in Section 3 furmulate the budget and deadline-constrained problem as an optimization problem for both fork&join and tree workflows. Then we describe our proposed algorithms for both workflows in Section 4, and give a numerical example in Section 5 to illustrate our algorithms. Finally, we conclude the paper in the last section.

2. RELATED WORK

Several grid workflow management systems with scheduling algorithms have been developed: Condor DAGMan [6], Pegasus [7], Kepler [15] and a Cloudbus workflow engine presented in [17]. Pegasus uses DAGMan to run the executable workflow. Kepler provides support for Web-service-based workflows. An actor-oriented design approach is used for executing and composing scientific application workflows. The computational components are called actors, and they are linked together to form a workflow. The Cloudbus workflow engine presented in [17] is a step closer toward scaling workflow applications on clouds using market-oriented computing.

In [27], Yu and Buyya provide a comprehensive taxonomy of workflow management systems based on workflow design, workflow scheduling, fault management, and data movement. They characterize and classify different approaches for building and executing workflows on Grids. Existing grid workflow systems highlighting key features and differences are also studied in the same paper.

In [24, 26] the authors presented a QoS-based workflow management system that uses a QoS-based workflow scheduling algorithm that minimizes the cost of execution while meeting the deadline. A Markov Decision Process approach to schedule sequential workflow task execution is used. Typical parameters that drive the scheduling decisions in such a QoS-based scenario include deadline (time) and budget (cost) [24, 18].

For instance, if the user requires to execute a given task at hand at a minimum execution cost, rather than using high-end but more expensive cloud resources, a policy for scheduling an application workflow that utilize local resources and then augment them with cheaper cloud resources should be in place. On the contrary, in order to achieve minimum execution time, a policy for scheduling workflows should

be used to always use high-end cluster and cloud resources, irrespective of costs. Deelman et al. [8] presented a simulation-based study on the costs involved when executing scientific application workflows using cloud services. They studied the cost performance trade-offs of different execution and resource provisioning plans. Yu and Buyya [25] also presented a budget constraint based workflow scheduling approach that minimizes the execution time while meeting a specified budget. A new type of genetic algorithm has also been developed, as the crossover and mutation operations of existing genetic algorithms focused on homogeneous and non reservation-enabled multiprocessor systems and therefore cannot be applied to the problem directly. The fitness function is developed to encourage the formation of the solutions to achieve the budget constraint and time minimization.

3. Problem Formulation

To uniformly formulate the problem for both workflows, we use double subscripts to denote each job in the workflow. For example, in the fork&join workflow, job J_{i_j} represents the jth job at stage i while in the tree workflow, J_{i_j} refers to the jth child job of job i for easy identification, which, of course, also has its own job number in the workflow. Note that in our notation, for the sake of consistent presentation we always use 0th child job to refer to the job itself.

3.1. *Job Model*

In our model, each job is associated with a set of machines, virtual or physical, that are provided by cloud service providers to execute this job. Each of these machines, depending on its performance or configuration, has independent (maybe different from each other) charge rates. The relationship between the performance and charge rate could be any function. In our case, we assume for job J_{i_j} $(0 \leq l \leq n_i)$, the available machines and the corresponding prices are defined as follows:

Table 1. Time-price table of job J_{i_j}

$$J_{i\_j} = \begin{bmatrix} t^1_{i\_j} & t^2_{i\_j} & ... & t^{m_{i\_j}}_{i\_j} \\ p^1_{i\_j} & p^2_{\_j} & ... & p^{m_{i\_j}}_{i\_j} \end{bmatrix} \tag{1}$$

.

Here, $t_u, 1 \leq u \leq m_{i\_j}$, represents the time to run job J_{i_j} on machine M_u whereas p_u represents the corresponding price for using that machine, and m_{i_j} is the total number of the machines that can run J_{i_j} (see Table 2 for more frequently used notations). Without loss of generality, we assume that the times are sorted in increasing order and the prices are in decreasing order. It is obvious that given two machines with the same run time for a job, no one will select the expensive one. Similarly, for two machines with the same price, no one will select the slow machine to run the job.

Let B_{i_j} denote the budget for job J_{i_j} and $T_{i\_j}(B_{i\_j})$ denote the shortest time to execute job J_{i_j}. $T_{i\_j}(B_{i\_j})$ is defined as

$$T_{i\_j}(B_{i\_j}) = t^u_{i\_j} \qquad p^{u+1}_{i\_j} < B_{i\_j} < p^{u-1}_{i\_j}. \tag{2}$$

Obviously, if $B_{i\_j} < p^{m_{i\_j}}_{i\_j}$, $T_{i\_j}(B_{i\_j}) = +\infty$.

Similarly, given D_{i_j} as the deadline for job J_{i_j}, we compute $C_{i\_j}(D_{i\_j})$, which denotes the lowest cost to execute job J_{i_j}, as

$$C_{i\_j}(D_{i\_j}) = p^u_{i\_j} \qquad t^u_{i\_j} \leq D_{i\_j} < t^{u+1}_{i\_j}. \tag{3}$$

Note that we require $t^1_{i\_j} \leq D_{i\_j} \leq t^{m_{i\_j}}_{i\_j}$. Otherwise $C_{i\_j}(D_{i\_j}) = +\infty$.

3.2. *Fork&Join Workflow*

Suppose a fork&join workflow consists of κ stages, each stage i $(0 \leq i \leq \kappa)$ having a collection of independent jobs, denoted as $J_{i\_1}, J_{i\_2}, ..., J_{i\_n_i}$, n_i is the job size of stage i, we formulate the problem based on budget and deadline constraints as follows:

3.2.1. *Budget Constraints*

The time to complete stage i with budget (monetary) B_i, denoted as $T_i(B_i)$, is defined as the time that allows the slowest job in that stage to be finished by using the given budget,

$$T_i(B_i) = \max_{\sum_{j\in[0,n_i]} B_{i\_j} = B_i} \{T_{i\_j}(B_{i\_j})\} \tag{4}$$

For fork&join, one stage cannot start until its immediately preceding stage has been finished, thus the total makespan with budget B to complete the workflow is defined as the sum of all stages' time, our goal is to minimize the time within the given budget B.

$$T(B) = \min_{\sum_{i\in[0,\kappa)} B_i = B} \sum_{i\in[0,\kappa)} T_i(B_i) \tag{5}$$

3.2.2. *Deadline Constraints*

Given a deadline D_i for stage i, the minimum cost (monetary) to finish stage i is

$$C_i(D_i) = \sum_{l\in[0,n_i]} C_{i\_j}(D_i) \tag{6}$$

where $C_{i\_j}(D_i)$ is the minimum cost to finish J_{i_j} in stage i within D_i. Again, we require $t^1_{i\_j} \leq D_i \leq t^{m_{i\_j}}_{i\_j}$. Otherwise $C_{i\_j}(D_i) = +\infty$. Finally, our optimization problem can be written as

$$C(D) = \min_{\sum_{i\in[1,k]} D_i = D} \sum_{i\in[0,\kappa]} C_i(D_i) \tag{7}$$

Table 2. Notation frequently used in model and algorithm descriptions

Symbol	*Meaning*
κ	the number of stages
n_i	the number of jobs in stage i $(0 < i \leq \kappa)$
B	the total budget for all jobs of κ stages
J_{i_j}	the jth job in stage i
$t^u_{i\_j}$	the time to run job J_{i_j} on machine M_u
$p^u_{i\_j}$	the cost rate for using M_u
m_{i_j}	the total number of the machines that can run J_{i_j}
B_{i_j}	the budget used by J_{i_j}
$T(B)$	the shortest time to finish the workflow given B
$T_i(B_i)$	the total time to finish stage i with budget B_i
$T_{i\_j}(B_{i\_j})$	the time to finish J_{i_j} with budget B_{i_j}
$T_{i\_0}(B_{i\_0})$	the time to finish job i when allocating budget B_{i_0} (tree)
D_i	the deadline to stage i
$C_{i\_j}(D_i)$	the minimum cost to finish job J_{i_j} in stage i within deadline D_i
$C(D)$	the minimum cost to finish the workflow within D
$C_{i\_0}(D_{i\_0})$	the minimum cost to finish job i within deadline D_{i_0} (tree)

3.3. *Tree Workflow*

Given a rooted tree workflow, we formulate the problem based on budget and deadline constraints as follows:

3.3.1. *Budget Constraints*

The time to complete a sub-tree rooted at job node i with budget (monetary) B_i, denoted as $T_i(B_i)$, is defined as the time that allows the slowest branch in that sub-tree to be finished by using the given budget,

$$T_i(B_i) = T_{i\_0}(B_{i\_0}) + \max_{\sum_{j \in Out(i)} B_{i\_j} = B_i - B_{i\_0}} \{T_{i\_j}(B_{i\_j})\} \tag{8}$$

where $Out(i)$ is job i's child job set. Again, $T_{i\_0}(B_{i\_0})$ represents the completion time of job i (not the sub-tree rooted at job i) when allocating budget B_{i_0}. Then the total makespan with budget B to complete the workflow can be simply computed by $T(B) = T_0(B)$ here job 0 is the rooted job in the tree workflow.

3.3.2. *Deadline Constraints*

Given a deadline D_i for a sub-tree rooted at job node i, the minimum cost (monetary) to finish this sub-tree is

$$C_i(D_i) = C_{i\_0}(D_{i\_0}) + \sum_{j \in Out(i)} C_j(D_i - D_{i\_0}) \tag{9}$$

where $C_{i\_j}(D_i - D_{i\_0})$ is the minimum cost to finish the sub-tree rooted at J_{i_j} within $D_i - D_{i\_0}$. By following the same arguments to Eq. (8), we have $C(D) = C_0(D)$ for a tree workflow rooted at job 0.

4. Optimal Scheduling Algorithms with Constraints

In this section, we present our optimal algorithms for both workflows under the budget and deadline constraints.

4.1. *Algorithms for Fork&Join*

4.1.1. *Optimization under Budget Constraints*

The proposed algorithm should have the capability of distributing the budget among the stages, and in each stage distributing the assigned budget to each constitute job in an optimal way. To this end, we design the algorithm in two steps:

(1) Given budget B_i for stage i, distribute the budget to all constitute jobs in such a way that $T_i(B_i)$ is minimum, that is

$$T_i(B_i) = \min_{\sum_{j \in [0,n_i]} \max_{B_{i\_j} = B_i} \{T_{i\_j}(B_{i\_j})\}} \tag{10}$$

Clearly, this computation is stage-wise and independent each other. Therefore they can be computed in parallel using κ machines.

(2) Given budget B for the workflow and the results in Eq. (10), optimize our goal Eq. (5).

For the first step, the n_i jobs in stage i are randomly associated with a label that forms a non-repetitive consecutive sequence of numbers starting from 1. Suppose the current budget is b, we use $T_i[j, b]$ to represent the minimum time to complete jobs indexed from j to n_j given budget b. To compute $T_i[j, b]$, we assign q units to the first job and the remaining $b - q$ units to the remaining jobs from $j + 1$ to n_i. Then for $0 \le i < \kappa$, $0 \le l \le n_i$, and $0 < b \le B_i$, we have the following recursions:

$$\begin{cases} T_i[j, b] = \min\limits_{0<q\le b} \{\max\{T_{i\_j}[q], T_i[j+1, b-q]\}\} \\ T_i[n_i, B_{i\_n_i}] = T_{i\_n_i}(B_{i\_n_i}) \qquad\qquad B_{i\_n_i} \le B_i \end{cases} \tag{11}$$

where the optimal solution to stage i can be found in $T_i[0, B_i]$.

Theorem 1. Given budget B_i for stage i having n_i jobs, Recursion (11) yields an optimal solution to the distribution of the budget B_i to all the n_i jobs in that stage within time $O(n_i B_i^2)$.

Proof. We prove this by induction on the number of jobs. Let the number of jobs, $n_i = 1$. Clearly, given budget b, the optimal solution is obtained by $T_i[n_i, b]$. Suppose there are n_i jobs, we consider jobs j and $j+1$. As an induction hypothesis, let $T_i[j+1, p]$ be an optimal solution to jobs from $j+1$ to n_i given budget p. We will show that $T_i[j, b]$ is an optimal solution to jobs from j to n_i under budget constraint $b \geq p$. In order to find the optimal distribution of the budget b among $n_i - j + 1$ jobs, we need to consider all the possibilities. To this end, we assign q units to the first job j and the remaining $b - q$ units to the remaining jobs from $j + 1$ to n_i, and allow q to be varied in the range of $(0, b]$. Clearly, the recursion chooses the minimum of all these, thus serving all the jobs from j to n_i with a minimum time.

Finally, at stage 1, since there are no more previous stages, the recursion (12) yields the optimal result $T_i[0, B_i]$ for the workflow. Since there are $O(n_i B_i)$ elements in the DP matrix (12). For each element, the computation complexity is at most $O(B_i)$ when $T_{i\_j}[q], 0 < q \leq b$ can be obtained in constant time (Table 1). Therefore, the total time complexity is $O(n_i B_i^2)$. Hence, the proof. □

Since all the κ stages can be computed in parallel, the total time complexity for the parallel pre-computation is $O(\max_{i \in [0,\kappa)} \{n_i B_i^2\})$.

Now we consider the second step. In this step, $p_{i\_j}^1, p_{i\_j}^2, ..., p_{i\_j}^{m_{i\_j}}$ are used to denote the prices to run job j on stage i on machine $1, 2, ..., m_{i\_j}$. Suppose $T_i[n_i, B_i]$, $\sum_{j=0}^{n_i} p_{i\_j}^{m_{i\_j}} \leq B_i \leq \sum_{j=0}^{n_i} p_{i\_j}^1$ are pre-computed, and there are a total of $\sum_{i=0}^{\kappa-1} \sum_{j=0}^{n_i} (p_{i\_j}^1 - p_{i\_j}^{m_{i\_j}})$ elements. Of course, instead of using the unit step width for q in Eq. (11), we can optimize the computation of the dynamic programming recursion by leveraging the discrete values of $p_{i\_j}^1, p_{i\_j}^2, ..., p_{i\_j}^{m_{i\_j}}$, which for all κ stages can totally reduce the number of elements to $\sum_{i=0}^{\kappa-1} \sum_{j=0}^{n_i} m_{i\_j}$. All of these pre-computed elements can be buffered for future use.

Given the results of Eq. (11) for all the κ stages, we try to obtain a dynamic programming recursion to compute the global optimal result. To this end, we use $T[i, b]$ to represent the minimum total time to complete stages indexed from i to κ when budget b is available, and have the following recursions ($0 < i \leq \kappa, 0 < b \leq B$):

$$T[i, b] = \begin{cases} \min\limits_{0<q\leq b} \{T_i[0, q] + T[i+1, b-q]\} & \text{if } i < \kappa \\ T_i[n_i, b] & \text{if } i = \kappa \end{cases} \tag{12}$$

where the optimal solution can be found in $T(1, B)$. The scheduling scheme can be reconstructed from $T(1, B)$ by recursively backtracking the DP matrix in (12) up to the initial budget distribution at stage κ which can phase by phase steer to the final optimal result. To this end, in addition to the time value, we can only store the budget q and the index of the previous stage (i.e., $T[i+1, b-q]$) in each cell of the matrix since given the budget for each stage, we can simply use Eq. (11) to

recompute the budget distribution.

Theorem 2. Given budget B for a κ-stage fork&join, each stage j having n_j jobs, Recursion (12) yields an optimal solution to the distribution of the budget B to all the κ stages within the pseudo-polynomial time complexity of $O(\kappa B^2)$.

Proof. We can reason about the correctness of this recursion by following the same arguments in Theorem 1. As such, the algorithm is pseudo-polynomial. □

Given the pseudo-polynomial time algorithm, we can see that if the bugdet is a small number, i.e., B is polynomially bounded in κ, then we would have a regular polynomial time

4.1.2. *Optimization under Deadline Constraints*

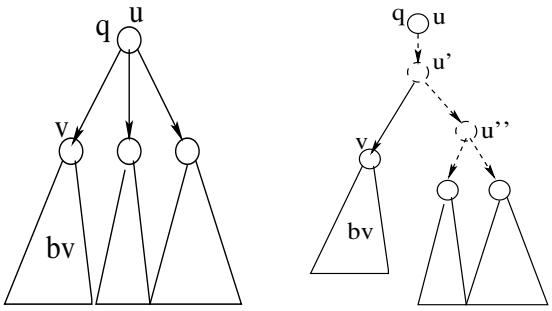
We partition the total deadline D into κ parts, denoted by $D_0, D_1, ..., D_{\kappa-1}$ such that $\sum_{0 \le i < \kappa} D_i \le D$. For a given D_i to stage i, we must ensure that all jobs in this stage can be finished within D_i, and to minimize the cost, we need to select the machine for each job on which the execution time of the job is the closest to D_i. Formally $C_{i\_j}(D_i) = p^u_{i\_j}$, $t^{u-1}_{i\_j} < D_i < t^{u+1}_{i\_j}$. Obviously, $C_{i\_j}(D_i)$ is the minimum cost to finish stage i within D_i. If stage i cannot be finished within D_i, $C_{i\_j}(D_i) = +\infty$. We then can compute Eq. (6).

By following the same arguments in Section 4.1.1, we can derive the optimal solution. However, this strategy is not efficient since in the previous case, the amount of allocated budget to each stage as well as its optimal distribution inside each stage cannot be computed in a simple way, which is different from the current case.

Alternatively, we can transform this problem into the standard MCKS problem by constructing κ classes in parallel, each corresponding to a stage in the workflow. The class i consists of a set of tuples (D_{i_j}, C_{i_j}) where $1 \le j \le \sum_{l \in [0,n_i]} p^l_{i\_j}$, representing the total minimum cost C_{i_j} for stage i under the given D_{i_j}. They are computed as follows,

(1) for each job J_{i_j} in stage i, gather its execution time on the candidate machines and put into set S;
(2) sort S in ascending order;
(3) for each element t_j in S, $D_{i\_j} \leftarrow t_j$ and then compute $C_{i\_j}(D_{i\_j})$ for each job j in stage i (this step can be further parallelized based on t_j);

The aim of the problem then becomes to picking up exactly one tuple from each class in order to minimize the total cost value of the pick, subject to the deadline constraint, which is a standard multiple-choice knapsack problem equivalent to Eq. (7). To optimize the computation, we can remove the tuple (D_{i_j}, C_{i_j}) from the class if $C_{i\_j} = +\infty$.



(a) Subtree (b) Transformed subtree

Fig. 2. A subtree and its transformed graph.

4.2. *Algorithms for Tree*

4.2.1. *Optimization Under Budget Constraints*

Our algorithm is directly derived from Eq. (8). Given budget B_i for a sub-tree rooted at job node i, our goal is to minimize the value of $T_i(B_i)$ when B_{i_0} is changed from $p_{i\_0}^{m_{i\_0}}$ to $p_{i\_0}^1$.

To this end, given budget b, we use $T[u, b]$ to represent the minimum time to complete the sub-tree rooted at node u. Then, for $0 < b \leq B$, we have

$$T[u,b] = \min_{0<q\leq b}\{T_{u\_0}(q) + \max_{\sum_{v\in Out(u)} b_v = b-q} T[v, b_v]\} \tag{13}$$

To solve recursion (13), by leveraging the idea in fork&join, we transform a general tree graph into a binary form by adding two dummy nodes u' and u", as shown in Fig. 2(a) and Fig. 2(b). Based on the transformed graph, we rewrite the recursion as follows,

$$\begin{cases} T[u,b] = \min_{0<q\leq b}\{T_{u\_0}(q) + S[u', b-q]\} \\ S[u', b-q] = \min_{0<p\leq b-q}\{T[v,p] + S[u'', b-q-p]\} \end{cases} \tag{14}$$

whereby, v is u's (also u''s) left most child job in the tree workflow. $S[u', b-q]$ is the shortest time to compute all the subtrees of node u with the remaining budget of $b-q$. First, the left-most child of u is assigned budget p, and then all its sibling subtrees which are hung on a dummy node use the remaining budget $b-q-p$. Dummy nodes are added to only ease the algorithm presentation; they are not budget consumers. This is the similar idea to use recursion (12) to compute $T(B)$ in Eq. (5) for the fork&join workflow. Therefore, the correctness of recursion (14) is not difficult to understand.

By following the arguments in Theorem 1, the time complexity of this algorithm is $O(nB^3)$ where n is the tree size given $0 < b \leq B$. As such, the algorithm is pseudo-polynomial.

```
---------------------------------
Stage: 0 has 3 jobs Max: 57.50(m)
---------------------------------
Job00: 4 machines
0: 12.50(m)     0.45($)
1: 37.50(m)     0.36($)
2: 45.00(m)     0.29($)
3: 48.75(m)     0.21($) *

Job01: 4 machines
0: 16.25(m)     0.47($)
1: 22.50(m)     0.36($)
2: 57.50(m)     0.15($) *
3: 61.25(m)     0.14($)

Job02: 4 machines
0: 20.00(m)     0.46($)
1: 36.25(m)     0.39($)
2: 38.75(m)     0.15($)
3: 50.0 (m)     0.14($) *
---------------------------------
Stage: 1 has 2 jobs Max: 45.00(m)
---------------------------------
Job10: 1 machines
0: 15.0 (m)     0.15($)

Job11: 4 machines
0: 18.75(m)    0.43($)
1: 41.25(m)    0.37($)
2: 45.00(m)    0.30($) *
3: 61.25(m)    0.17($)
---------------------------------
Stage: 2 has 2 jobs Max: 58.75(m)
---------------------------------
Job20: 4 machines
0: 13.75(m)    0.45($)
1: 17.50(m)    0.44($)
2: 38.75(m)    0.39($)
3: 58.75(m)    0.10($) *

Job21: 4 machines
0: 18.75(m)    0.44($)
1: 36.25(m)    0.20($)
2: 48.75(m)    0.13($)
3: 57.5 (m)    0.11($) *
---------------------------------
```

Fig. 3. A 3-stage fork&join workflow. For each job, there is a set of candidate machines that can run the job, and their time-price information is shown, the left column is the execution time, and the right column is the corresponding service cost. '*' marks a scheduling scheme.

4.2.2. *Optimization Under Deadline Constraints*

By comparing Eq. (9) with Eq. (8), we can find the optimization under deadline constraints is relatively easy as the deadline time is not necessarily distributed among the subtrees of the rooted job node. Instead, all the substrees share the same deadline $D_i - D_{i\_0}$ if D_{i_0} is assigned to the root node. Given this consideration, we have the following algorithm,

$$C_i(D_i) = \min_{t^1_{i\_0} \le D_{i\_0} \le t^{m_{i\_0}}_{i\_0}} \{C_{i\_0}(D_{i\_0}) + \sum_{j \in Out(i)} C_j(D_i - D_{i\_0})\}, \tag{15}$$

and the opitmal cost of scheduling the tree workflow with deadline D can be found at $C_0(D)$.

5. Numerical Examples

To illustrate the proposed algorithms, we implemented them in Java and applied it to an example shown in Fig. 3 where a 3-stage fork&join workflow, each with 3 or 2 jobs, is scheduled in a cloud platform. We select a fork&join workflow as the example because the optimal scheduling of this workflow contains the core algorithm (i.e., Eq. (12)) in this paper, which has been leveraged by the solutions to both the studied workflows. Thus, for illustration purposes, examining the algorithm for the fork&join workflow is sufficient.

In our fork&join example, for each job in a stage, there is a set of candidate machines that can run the job with different service rates and performance. The time-price information is also summarized in Fig. 3.

Table 3. DP matrix of the optimal budget distribution for Fig. 3.
The first row is the budget while others are execution times.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3
stage0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	177.5	**161.25**
stage1	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	120.0	103.75	77.5	77.5	76.25	57.5	55.0
stage2	$+\infty$	$+\infty$	58.75	58.75	57.5	38.75	36.25	36.25	18.75	18.75	18.75	18.75	18.75

Given budget \$1.3, we can compute the DP matrix of Recursion (12) by using the results in (11). The matrix is shown in Table 3 where the optimal computation time is 161.25(m), and $+\infty$ indicates that the budget is insufficient for the computation. The scheduling scheme can be reconstructed by backtracking the DP matrix of Recursion (12) when more bookkeeping information is stored during the computation in each cell of the matrix. With such information, we can obtain a schedule with length of 161.25(m) that is also marked by $'*'$ in Fig. 3. For instance, `Job00` is allocated to `Machine3` by paying `$0.21` for completion within `48.75(m)`. Note that the rate of increase in the DP computation is \$0.1, which is higher than the rate of increase using a real service rates model. This mismatch would lead to the loss of optimality in the final result. A major reason that we adopt this increase rate in this example is to reduce the size of the otherwise large DP matrix. On the other hand, changing the rate of increase makes a trade-off between the optimality of the results and computation overhead, which is also very important, especially when the budget and workflow size are large. Obviously, when the rate of the increase matches the real service rate increase, Recursion (12) returns the optimal results.

6. Conclusions

In this paper, we studied two practical constraints: budget and deadline, in optimal scheduling of two well-structured workflows in scientific computation, fork&join and tree, on a set of (virtual) machines in the cloud. Each of the two proposed problems has its own scheduling goal. We presented optimal algorithms for both workflows based on dynamic programming techniques to address each respective problem with pseudo-polynomial time complexities whereby we can conduct further research to find the possible PTAS, or even FPTAS (say using scaling technique) which could be more practical for the problems.

Acknowledgement

The authors would like to thank anonymous reviewers who gave valuable suggestion that has helped to improve the quality of the manuscript. This research is sponsored in part by funding provided to the Centre for Advanced Studies - Atlantic from IBM

and the Atlantic Canada Opportunities Agency through the Atlantic Innovation Fund.

References

[1] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, Mei-Hui Su, and K. Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10, Nov. 2008.

[2] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. WIEN2k: An augmented plane wave plus local orbitals program for calculating crystal properties. Technical report, Institute of Physical and Theoretical Chemistry, Vienna University of Technology, 2001.

[3] Jens Brüning and Peter Forbrig. TTMS: A task tree based workflow management system. In Terry Halpin, Selmin Nurcan, John Krogstie, Pnina Soffer, Erik Proper, Rainer Schmidt, and Ilia Bider, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *Lecture Notes in Business Information Processing*, pages 186–200. Springer Berlin Heidelberg, 2011.

[4] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010.

[5] Eddy Caron, Frédéric Desprez, Adrian Muresan, and Frédéric Suter. Budget constrained resource allocation for non-deterministic workflows on an iaas cloud. In *Proceedings of the 12th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'12, pages 186–201, 2012.

[6] Condor Team. `http://www.cs.wisc.edu/condor/dagman`, access date: 09/10/2012.

[7] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[8] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008.

[9] Eslam Elnikety, Tamer Elsayed, and Hany E. Ramadan. ihadoop: Asynchronous iterations for mapreduce. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 81–90, 2011.

[10] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640 –645, dec 2008.

[11] M. Jacquelin, L. Marchal, Y. Robert, and B. Uar. On optimal tree traversals for sparse matrix factorization. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 556–567, May 2011.

[12] Gideon Juve and Ewa Deelman. Scientific workflows and clouds. *Crossroads*, 16(3):14–18, Mar. 2010.

[13] Gideon Juve, Ewa Deelman, G. Bruce Berriman, Benjamin P. Berman, and Philip Maechling. An evaluation of the cost and performance of scientific workflows on amazon ec2. *J. Grid Comput.*, 10(1):5–21, mar 2012.

[14] Jimmy Lin and Michael Schatz. Design patterns for efficient graph algorithms in mapreduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 78–85, New York, NY, USA, 2010. ACM.

[15] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006.

[16] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, 2010.

[17] Suraj Pandey, William Voorsluys, Mustafizur Rahman, Rajkumar Buyya, James E. Dobson, and Kenneth Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurr. Comput. : Pract. Exper.*, 21(16):2118–2139, November 2009.

[18] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 400–407, 2010.

[19] David Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83:394–410, 1994.

[20] R. Schreiber. A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Softwoare*, 8(3):256–276, 1982.

[21] Dieter Schuller, Andr Miede, Julian Eckert, Ulrich Lampe, Apostolos Papageorgiou, and Ralf Steinmetz. Qos-based optimization of service compositions for complex workflows. In PaulP. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 641–648. Springer Berlin Heidelberg, 2010.

[22] Prabhakant Sinha and Andris A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):pp. 503–515, 1979.

[23] J. Wang, H. Kuehl, and M.D. Sacchi. Least-squares wave-equation AVP imaging of 3D common azimuth data. In *Proceedings of the 73rd Annual International Meeting, Society of Exploration Geophysicists*, 2003.

[24] J. Yu and R Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Sci. Program.*, 14(3,4):217–230, December 2006.

[25] Jia Yu and R. Buyya. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Workflows in Support of Large-Scale Science, 2006. WORKS '06. Workshop on*, pages 1–10, 2006.

[26] Jia Yu and R. Buyya. Qos-based scheduling of workflows on global grids. In *Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 1–10, 2007.

[27] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Record*, 34(3):44–49, 2005. Special section on scientific workflows.

[28] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, AINA '12, pages 534–541, 2012.

[29] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. Priter: A distributed framework for prioritizing iterative computations. *Parallel and Distributed Systems,*

IEEE Transactions on, 24(9):1884–1893, Sept 2013.