Performance Optimization of Hadoop Workflows in Public Clouds through Adaptive Task Partitioning

Tong Shu[†] and Chase Q. Wu[†‡]
[†]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
[‡]School of Information Science and Technology, Northwest University, Xi'an, Shaanxi 710127, China
Email: {ts372, chase.wu}@njit.edu

***Abstract*—Cloud computing provides a cost-effective computing platform for big data workflows where moldable parallel computing models such as MapReduce are widely applied to meet stringent performance requirements. The granularity of task partitioning in each moldable job has a significant impact on workflow completion time and financial cost. We investigate the properties of moldable jobs and design a big-data workflow mapping model, based on which, we formulate a workflow mapping problem to minimize workflow makespan under a budget constraint in public clouds. We show this problem to be strongly NP-complete and design i) a fully polynomial-time approximation scheme (FPTAS) for a special case with a pipeline-structured workflow executed on virtual machines in a single class, and ii) a heuristic for a generalized problem with an arbitrary directed acyclic graph-structured workflow executed on virtual machines in multiple classes. The performance superiority of the proposed solution is illustrated by extensive simulation-based results in Hadoop/YARN in comparison with existing workflow mapping models and algorithms.**

I. Introduction

Next-generation applications in science, industry, and business domains are producing colossal amounts of data, now commonly termed as "big data", which must be analyzed in a timely manner for knowledge discovery and technological innovation. Among many existing solutions, data-intensive computing workflows comprised of MapReduce jobs have become an indispensable technique for big data analytics.

In recent years, an increasing number of big data workflows have migrated to clouds, which has reaped the benefit of resource virtualization but meanwhile has also brought many new challenges for workflow execution and performance optimization [18]. Cloud computing makes computing a utility such that one pays for only those cloud resources that are truly needed and used [12]. Hence, to support cost-effective execution of big data workflows in clouds, researchers are now facing the challenge of reducing financial cost in addition to meeting traditional performance optimization goals [18].

Parallel computing is generally categorized into three classes with flexibility from low to high: i) rigid jobs exemplified by multi-threaded programs running on a fixed number of processors, ii) moldable jobs exemplified by MapReduce programs running on any number of processors decided prior to execution, and iii) malleable jobs running on a variable number of processors at runtime [4]. Most existing efforts on workflow optimization consider serial or rigid jobs with execution dependencies. In big data systems such as Hadoop, there arises a new challenge to optimize the mapping of moldable parallel jobs, each of which has a variable number of independent homogenous tasks. A moldable job typically follows a performance model where the workload of each component task decreases while the total workload, which determines the financial cost, increases as the number of allotted processors increases [7]. This performance model has been validated by many real-life parallel programs running on disparate high-performance computing platforms and will serve as a base of our workflow mapping solution for performance optimization of big data workflows.

In this paper, we design a big-data workflow mapping model, where a job scheduler adaptively partitions each MapReduce job into a certain number of homogeneous tasks and executes them on a selected set of VM instances in a single class of the same processing speed. Based on this mapping model, we construct analytical cost models in a combination of a workflow engine and a Hadoop/YARN resource manager and formulate a MapReduce workflow mapping problem to minimize workflow makespan under a given budget constraint in public clouds. We show this problem to be strongly NP-complete, and design a fully polynomial-time approximation scheme (FPTAS) for a special case with a chain of linearly arranged jobs on VMs in a single class and a heuristic for a generalized problem with an arbitrary directed acyclic graph (DAG)-structured workflow on VMs in multiple classes. The performance superiority of the proposed solution in terms of workflow makespan and financial cost is illustrated by extensive simulation results in Hadoop/YARN in comparison with existing algorithms.

The rest of the paper is organized as follows. Section II provides a survey of related work. Section III formulates a big data workflow mapping problem. We design an FPTAS for a special case with a pipeline-structured workflow in Section IV, and a heuristic for a generalized problem in Section V. Section VI presents the performance evaluation.

II. Related Work

We conduct a survey of related work on budget-constrained workflow performance optimization and workflow mapping with malleable jobs.

A. Budget-constrained Workflow Performance Optimization

Several recent efforts address workflow scheduling that takes both cost and delay into consideration in cloud environments [19], [1], [10]. Rodriguez *et al.* proposed a combined resource provisioning and scheduling strategy for executing scientific workflows in IaaS clouds to minimize the overall execution cost while meeting a user-defined deadline [14]. Wu *et al.* formulated a job scheduling problem to minimize the end-to-end delay of a workflow under a user-specified financial constraint in clouds and designed a heuristic solution [18]. The above work is focused on performance optimization in clouds, without considering big data workflows in Hadoop.

Research efforts on scheduling a batch of MapReduce jobs under a budget constraint in Hadoop systems are still quite limited. Sandholm *et al.* proposed the dynamic priority task scheduling (DPSS) algorithm for heterogeneous Hadoop clusters [15]. Our work differs from DPSS in two aspects: i) We consider sufficient capacities in clouds with different VM types to minimize the makespan within a given budget, while DPSS allows dynamic capacity distribution across concurrent users based on user preferences. ii) We aim to optimize the mapping of a workflow with MapReduce jobs from a global perspective, while DPSS attempts to optimize the budget on a per-job basis by allowing users to adjust their spending over time. Wang *et al.* modeled a batch of MapReduce jobs as a multi-stage fork-and-join workflow with no precedence in each stage and proposed one pseudo-polynomial optimal solution and two heuristics for task-level scheduling to minimize the maximum completion time under a budget constraint on heterogeneous VMs [17]. In their mapping model, each task in a job is mapped onto a different VM instance; while in our mapping model, each job with adaptive task partitioning is mapped onto a set of shared VM instances with identical processing speed. In their algorithms, the number of stages has a significant impact on the optimality and time complexity; while in our algorithm, we do not divide a workflow into stages for performance improvement. Huang *et al.* designed a system, Cumulon, to help users rapidly develop and strategically deploy matrix-based big-data analysis programs in clouds according to time/budget constraints [8]. Different from Cumulon, our work is focused on the design of big-data workflow mapping algorithms in Hadoop/YARN.

B. *Workflow Mapping with Malleable Jobs*

Several efforts have been made to minimize the maximum or total completion time of malleable jobs with execution precedences [13], [11]. Jansen *et al.* proposed a two-phase approximation algorithm with an approximation ratio of 3.291919 for scheduling workflows with malleable jobs [9]. Chen *et al.* considered the scheduling of malleable jobs under general precedence constraints to find a minimum makespan, assuming that i) the processing time and the workload of a malleable job are non-increasing and non-decreasing, respectively, as the number of assigned processors increases, and ii) the workload function is convex with respect to the processing time. They proposed a polynomial-time approximation algorithm with an approximation ratio of 3.4142, which leads to an approximation ratio of 2.9549 when the processing time is strictly decreasing as the number of assigned processors increases [2]. The above work is focused on the theoretical analysis of malleable computing models. In this paper, we attempt to minimize the makespan of a workflow with moldable jobs.

III. Problem Formulation

We construct rigorous cost models and formulate a budget-constrained workflow mapping problem for makespan minimization.

A. *Cost Models*

1) *Cloud Model*

We consider a set Y of available VM types, partitioned into multiple classes $\{C_i\}$, such as general-purpose VMs, computation-optimized VMs, memory-optimized VMs, storage-optimized VMs, and GPU-based VMs in Amazon EC2. The VM types in the same class have the same computer architecture (i.e. identical processing speed) with different specifications, and the VM types in different classes have different computer architectures. Each VM type y_j in class C_i is associated with performance- and price-related attributes (s_i, n_j, m_j, p_j), where i) s_i denotes the processing speed of a CPU core in class C_i, ii) n_j denotes the number of homogeneous CPU cores of VM type y_j, iii) m_j denotes the memory size of VM type y_j, and iv) p_j denotes the price for using a VM instance of type y_j per time unit. In general, the price is a linear function with respect to the capacity of the VM types in a single class, and the capacity of a VM type is α_j ($\alpha_j > 1, \alpha_j \in \mathbb{Z}$) times that of the next lower VM type, as in Amazon EC2 pricing model.

2) *Workflow Model*

We consider a user request in the form of a workflow $f(G, B)$, which specifies a workflow structure G and a budget B for the entire workflow execution. The workflow structure is defined as a DAG $G(V, A)$, where each vertex $v_k \in V$ represents a component job, and each directed edge $a_{k,k'} \in A$ between job v_k and job $v_{k'}$ denotes an execution dependency, i.e. the actual finish time (AFT) t_k^F of job v_k must not be later than the actual start time (AST) $t_{k'}^S$ of job $v_{k'}$. The completion time of the workflow is denoted as t^C. We consider the map and reduce phases of each MapReduce job as two component jobs connected via an execution dependency edge.

3) *MapReduce Job Model*

We consider a component job v_k that contains a set of parallel map (or reduce) tasks, each of which requires a memory of size m_k and spends a percentage $u_{i,k}$ of time executing CPU-burst instructions on a CPU core of a VM in class C_i. In job v_k, as the number L_k of parallel tasks in v_k increases, the total workload $w_k(L_k)$ of all tasks would increase while the workload $w_{k,l}(L_k) = w_k(L_k)/L_k$ of each task $r_{k,l}$ would decrease.

In Hadoop/YARN, a portion of input data to be processed by one map task is called a split. The largest number of splits of a MapReduce job is typically equal to the number of data blocks (the data unit in HDFS) in the input data (i.e., a map task processes one data block), which is decided by the volume of the entire dataset divided by the data block size of HDFS. In general, the number of reduce tasks is much less than the number of reduce keys in a MapReduce job. Hence, the maximum number L'_k of tasks in job v_k is limited by the number of input data blocks of the MapReduce job. We assume that each task is assigned onto a different CPU core.

4) *Time Model*

Since contiguous tasks can time-share a common VM instance over different periods, the total VM initialization time for large-scale workflows could be negligible. Since our work targets big data applications in cloud environments in a single data center where physical machines are interconnected via high-speed links, we do not specifically consider data transfer time. Hence, the time cost of a task $r_{k,l}$ running on a VM instance of a type in class C_i can be simplified as the execution time of $r_{k,l}$ on a VM in C_i, i.e. $t_{i,k,l} = w_{k,l}(L_k)/(u_{i,k} \cdot s_i)$, and so is the time cost $t_{i,k}(L_k)$ of job v_k if all L_k tasks start to run on VM instances in class C_i at the same time.

5) *Financial Cost Model*

Data transfer within the same data center is typically free of charge. For example, in the Amazon pricing scheme, there is no data transfer charge between Amazon EC2 and S3 within the same region. Furthermore, public clouds support scalable storage by enabling a user to create an Amazon EBS volume and attach it to a VM instance, so the storage size is independent of the VM type. In addition, the price for storage is counted in unit of months, so the financial cost of storage during task execution may be ignored. Hence, the expense of executing a job v_k with L_k tasks on VM instances of types in class C_i can be simplified as the financial cost of the active VM instances, on which L_k tasks are running during period $t_{i,k}(L_k)$, i.e. $e_{i,k}(L_k) = t_{i,k}(L_k) \cdot \sum_{y_j \in C_i}(p_j h_{j,k})$, where $h_{j,k}$ is the number of VM instances of type y_j in class C_i assigned to job v_k. Given L_k tasks in job v_k and a single class of VM types, we can find the least expensive combination of VM instances in linear time, which are powerful enough to run L_k tasks simultaneously. Thus, the expense $e_{i,k}(L_k)$ of job v_k mentioned below denotes the minimum expense of job v_k with L_k tasks on VM instances in class C_i.

6) *Mapping Function*

We define a workflow mapping function as $\mathfrak{M} : \{v_k \xrightarrow[L_k]{[t_k^S, t_k^F]} C_i, \forall v_k \in V, \exists L_k \in [1, L'_k], \exists C_i \subset Y, \exists [t_k^S, t_k^F] \subset T\}$, which denotes that the k-th job is partitioned into L_k tasks and mapped onto a set of VM instances of types in the i-th class from time t_k^S to time t_k^F. The domain of this mapping function covers all possible combinations of a set V of moldable jobs of the workflow, a set of VM classes, and a time period T of workflow execution.

B. *Problem Definition*

We formulate a Budget-Constrained Workflow Mapping problem with Moldable jobs for Makespan Minimization in public Clouds, referred to as BCWM4C, as follows.

Definition 1. ***BCWM4C***: *Given a set Y of VM types divided into classes $\{C_i\}$, and a workflow request $f(G(V,A),B)$, where each job v_k has a set $\{w_k(L_k)|L_k = 1, 2, \ldots, L'_k\}$ of workloads corresponding to different degrees of parallelism, and each task in job v_k has a memory demand m_k and spends $u_{i,k}$ percent of time executing CPU-burst instructions on a VM in class C_i, we wish to find a mapping function $\mathfrak{M} : (V, Y, T) \rightarrow \{v_k \xrightarrow[L_k]{[t_k^S, t_k^F]} C_i\}$ to minimize makespan:*

$$\min_{\mathfrak{M}} t^C,$$

subject to the budget and precedence constraints:

$$\sum\nolimits_{v_k \in V} e_k \leq B,$$

$$t_k^F \leq t_{k'}^S, \forall a_{k,k'} \in A.$$

TABLE I

NOTATIONS USED IN THE COST MODELS.

Notations	Definitions
$Y = \bigcup_i C_i$	a set of VM types divided into classes $\{C_i\}$
$y_j(s_i, n_j, m_j, p_j)$	the j-th VM type of price p_j, equipped with a memory of size m_i and n_j CPU cores of speed s_i
$f(G(V,A), B)$	a workflow request consisting of a workflow structure of a DAG $G(V,A)$ and a budget B
v_k, $r_{k,l}$	the k-th component job in a workflow and the l-th task in job v_k
$a_{k,k'}$	the directed edge from job v_k to job $v_{k'}$
t_k^S, t_k^F	the actual start and finish time of job v_k
t^C	the completion time of a workflow
$u_{i,k}$	the percentage of execution time for CPU-burst instructions in job v_k on a VM instance of a type in class C_i
m_k	the memory demand per task in job v_k
$w_k(L)$	the workload of job v_k partitioned into L tasks
$w_{k,l}(L)$	the workload of task $r_{k,l}$ in job v_k with L tasks
L_k, L'_k	the number and the maximum possible number of tasks in job v_k
$t_{i,k,l}$	the execution time of task $r_{k,l}$ running on a VM in class C_i
$t_{i,k}(L)$ $e_{i,k}(L)$	the execution time and minimun expense of job v_k with L tasks running in parallel on VM instances of types in class C_i
$h_{j,k}$	the number of VM instances of type y_j assigned to job v_k

C. *Complexity Analysis*

We first consider a special case of BCWM4C as follows: given a VM type of price p with sufficient memory and 5 CPU cores of speed s, and K independent serial jobs $\{v_k\}$ with CPU-burst workload w_k, does there exist a feasible non-preemptive scheduling scheme under a budget constraint B such that the makespan is no more than $T = B/p$? This special case has been proved to be strongly NP-hard in [5], so is the general BCWM4C problem, which, with a polynomially bounded objective function, has no FPTAS unless $P = NP$ [16].

IV. A Special Case: a Pipeline-structured Workflow on VMs in a single class

We start with a special case with a pipeline-structured workflow running on VM instances of types in a single class (PWSC), which is also NP-complete, and design an FPTAS. The BCWM4C-PWSC problem is formally defined as follows.

Definition 2. ***BCWM4C-PWSC***: *Given J VM types $\{y_j(s, n_j, m_j, p_j)\}$ in a single class with increasing capacity, and a workflow $G(V,A)$ containing a chain of K jobs, where each job v_k has a workload list $\{w_k(L_k)|L_k = 1, 2, \ldots, L'_k\}$, and each task in job v_k has a memory demand m_k ($\leq m_j$) and spends u_k percent of time executing CPU-burst instructions, does there exist a feasible mapping scheme such that the expense and the makespan are no more than a budget B and a bound T, respectively?*

A. *Complexity Analysis*

We prove BCWM4C-PWSC to be NP-complete by reducing the NP-complete two-choice knapsack problem (TCKP) to it.

Definition 3. ***Two-Choice Knapsack***: *Given K classes $S_1, S_2, \ldots, S_K$ of items to pack in a knapsack of capacity Q, where each class has exactly two items $d_{k,l} \in S_k$*

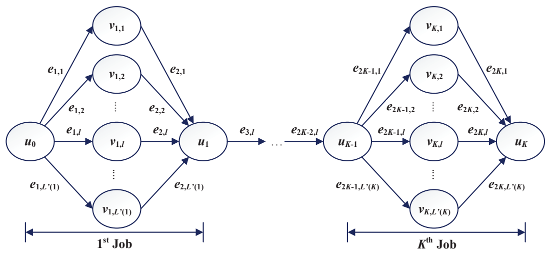


Fig. 1. A constructed network corresponding to a workflow with a pipeline structure, where $L'(1) = L'_1$ and $L'(K) = L'_K$.

Algorithm 1: PMMM()

```
1: Construct a DAG $\mathbb{G}(\mathbb{V}, \mathbb{E})$ for a given pipeline-structured workflow as
   shown in Fig. 1, and assign cost $e_k(l)$ and length $t_k(l)$ to edge $e_{2k-1,l}$
   and zero cost and zero length to edge $e_{2k,l}$;
2: Use FPTAS in [6] to find the shortest path from $u_0$ to $u_K$ under cost
   constraint $B$ with approximate rate $\epsilon_1$ and convert it to mapping scheme
```

Algorithm 2: PMEM()

```
1: Construct a DAG $\mathbb{G}(\mathbb{V}, \mathbb{E})$ for a given pipeline-structured workflow as
   shown in Fig. 1, and assign cost $t_k(l)$ and length $e_k(l)$ to edge $e_{2k-1,l}$
   and zero cost and zero length to edge $e_{2k,l}$;
2: Use FPTAS in [3] to find the shortest path from $u_0$ to $u_K$ under cost
   constraint $T$ with approximate rate $\epsilon_2$ and convert it to mapping scheme
```

($l \in \{1, 2\}, k = 1, 2, \ldots, K$), each of which has a value $o_{k,l}$ and a weight $q_{k,l}$, is there a choice of exactly one item from each class such that the total value is no less than O and the total weight does not exceed capacity Q?

The 0-1 knapsack problem is a special case of TCKP where we put each item from the 0-1 knapsack problem and a dummy item with zero value and zero weight in the same class. Since the knapsack problem is NP-hard, so is TCKP.

Theorem 1. *BCWM4C-PWSC is NP-complete.*

Proof. Obviously, BCWM4C-PWSC $\in NP$. We prove BCWM4C-PWSC to be NP-hard by reducing TCKP to it.

Let $(\{S_k(o_{k,1}, q_{k,1}, o_{k,2}, q_{k,2})|1 \leq k \leq K\}, O, Q)$ be an arbitrary instance of TCKP. Without loss of generality, we assume that $o_{k,1} > o_{k,2}$, and $q_{k,1} > q_{k,2} > 0$. If $q_{k,1} \leq q_{k,2}$, $d_{k,1}$ would always be picked. If $q_{k,2} = 0$, we can always add $\tau > 0$ to $q_{k,1}$, $q_{k,2}$, and Q such that $q_{k,2} > 0$.

We construct an instance of BCWM4C-PWSC as follows. Let $n_1 = 1$, $n_2 = 2$, $m_2 = 2m_1$, $p_1 = (o_{k,1} - o_{k,2})/(2q_{k,2} - q_{k,1})$, $p_2 = 2p_1$, $v_k = S_k$, $L'_k = 2$, $w_k(1) = q_{k,1}u_k s$, $w_k(2) = 2q_{k,2}u_k s$, $T = Q$, and $B = \sum_{1 \leq k \leq K} O_k - O$, where $O_k = (2q_{k,2}o_{k,1} - q_{k,1}o_{k,2})/(2q_{k,2} - q_{k,1})$. It is obvious that this construction process can be done in polynomial time.

Then, if job v_k only has one task, the task is mapped onto one VM instance of type y_1, so its execution time is $t_k(1) = w_k(1)/(u_k s) = q_{k,1}$, and its expense is $e_k(1) = p_1 t_k(1) = O_k - o_{k,1}$. If job v_j has two tasks, the execution time of each task is $t_k(2) = w_k(2)/(2u_k s) = q_{k,2}$, and the expense of job v_k is $e_k(2) = p_2 t_k(2) = O_k - o_{k,2}$. Obviously, two tasks in job v_k are mapped onto two VMs of type y_1 or one VM of type y_2 simultaneously.

Therefore, if the answer to the given instance of TCKP is YES (or NO), the answer to the constructed instance of BCWM4C-PWSC is also YES (or NO). Proof ends. □

B. Approximation Algorithm

We design an FPTAS to solve BCWM4C-PWSC by reducing BCWM4C-PWSC to the restricted shortest path (RSP) problem, which is solvable with an FPTAS.

Given an instance of BCWM4C-PWSC, we construct an instance of RSP according to the pipeline as follows. As illustrated in Fig. 1, the network graph G consists of $\mathbb{V} = \{v_{k,l}|k = 1, \ldots, K, l = 1, \ldots, L'_k\} \cup \{u_0, u_k|k = 1, \ldots, K\}$ with a source u_0 and a destination u_k, and $\mathbb{E} = \{e_{2k-1,l}, e_{2k,l}|k = 1, \ldots, K, l = 1, \ldots, L'_k\}$, where $e_{2k-1,l} = (u_{k-1}, v_{k,l})$ and $e_{2k,l} = (v_{k,l}, u_k)$. Then, we calculate the execution time of job v_k with l tasks as $t_k(l) = w_k(l)/(l \cdot s \cdot u_k)$, and accordingly its expense as $e_k(l) = t_k(l)p_k(l)$, where $p_k(l)$ is the least expensive price for executing l tasks in v_k in parallel.

Based on the same decision version of BCWM4C-PWSC in Definition 2, we provide an FPTAS to each of its two different optimization problems as follows:

1) To minimize makespan under a budget constraint

In Alg. 1, we assign the cost $c(e)$ and length $h(e)$ of each edge $e \in \mathbb{E}$ as $c(e_{2k-1,l}) = e_k(l)$, $l(e_{2k-1,l}) = t_k(l)$, and $c(e_{2k,l}) = l(e_{2k,l}) = 0$, and set the cost constraint on a path from u_0 to u_K to be budget B. As a result, the shortest length in RSP is exactly the minimum makespan in BCWM4C-PWSC, and if $v_{k,l}$ is on the solution path to RSP, the k-th job has l tasks. If the FPTAS in [6] is used to solve RSP, BCWM4C-PWSC finds a feasible solution within the shortest makespan multiplied by $(1+\epsilon_1)$ in time $O(K^2 L'^2/\epsilon_1)$, where $L' = \max_{1 \leq k \leq K} L'_k$.

2) To minimize expense under a makespan constraint

In Alg. 2, we swap the cost and length of each edge in Alg. 1, and set the path cost constraint to be makespan bound T. Similarly, the shortest length in RSP is exactly the minimum expense in BCWM4C-PWSC. If the FPTAS in [3] is used to solve RSP, BCWM4C-PWSC finds a solution with the least expense under the makespan constraint multiplied by $(1+\epsilon_2)$ in time $O((K^2 L')/\epsilon_2)$, thanks to the special topology in Fig. 1.

V. Algorithm Design for Arbitrary Workflows on VMs in Different Classes

We consider BCWM4C with a DAG-structured workflow on heterogeneous VM types in different classes and design a heuristic algorithm, big-data adaptive workflow mapping (BAWM), for minimum end-to-end delay.

A. An Overview of BAWM

For the sake of clarification, we list some variables and functions used in BAWM in Table II. Each job v_k is associated with a set of pairs of the number $L_{k,n}$ of tasks ($L_{k,n} \in [1, L'_k]$) and the class $C_{k,n}$ of assigned VM types ($C_{k,n} \in \{C_i\}$). Each pair corresponds to certain execution time $t_{k,n} = t_k(L_{k,n}, C_{k,n})$

TABLE II
VARIABLES AND FUNCTIONS USED IN ALGORITHM DESIGN.

Notations	Definitions
$e_{\min}$	the smallest allowable budget for a feasible workflow mapping scheme
$e_{\max}$	the smallest sufficient budget for the fastest execution of all jobs
$(t, e, optIdx) \leftarrow save(f, tet, optIdx)$	Reclaim unnecessary budget allocation for workflow f without makespan increase and then return makespan t, expense e, and the option indices $optIdx$ of all jobs in the latest mapping
$t \leftarrow calMS(f, tet, optIdx)$	Calculate the makespan of workflow f based on the TETs tet and option indices $optIdx$ of all jobs
$e \leftarrow calExp(f, tet, optIdx)$	Calculate the expense e of workflow f based on the TETs tet and option indices $optIdx$ of all jobs

TABLE III
TIME-EXPENSE TABLE tet_k OF JOB v_k.

$$\begin{array}{ccccccccc}
t_{k,1} & < & t_{k,2} & < & \cdots & < & t_{k,n} & < & \cdots & < & t_{k,N} \\
e_{k,1} & > & e_{k,2} & > & \cdots & > & e_{k,n} & > & \cdots & > & e_{k,N} \\
L_{k,1} & & L_{k,2} & & \cdots & & L_{k,n} & & \cdots & & L_{k,N} \\
C_{k,1} & & C_{k,2} & & \cdots & & C_{k,n} & & \cdots & & C_{k,N}
\end{array}$$

Algorithm 3: BAWM()

```
Input: A workflow $f(G(V,A),B)$ and a set $Y = \bigcup_i C_i$ of VM types
Output: a makespan $t$ and a remaining budget $b$
1: Build TETs $tet$ for all jobs in $V$ according to a set $V$ of jobs and a set
   of classes $\{C_i\}$ of VM types;
2: Find the minimum expense as $e_{\min}$ when each job selects the least
   expensive execution option and the maximum expense as $e_{\max}$ when
   each job selects the fastest execution option;
3: if $B < e_{\min}$ then
4:     Exit with an error.
5: if $f$ is a pipeline then
6:     Use Alg. 1 to compute the option index $optIdx[k]$ of each job $v_k$;
7:     return $(calMS(f, tet, optIdx), B - calExp(f, tet, optIdx))$;
8: for all $v_k \in V$ do
9:     $optIdx[k] \leftarrow$ the index of the first (fastest) option in $tet_k$;
10: $(t, e, optIdx) \leftarrow save(f, tet, optIdx)$;
11: if $e \leq B$ then
12:     return $(t, B - e)$;
13: if $B \leq e_{\min} + \delta \cdot (e_{\max} - e_{\min})|_{\delta=0.07}$ then
14:     for all $v_k \in V$ do
15:         $optIdx[k] \leftarrow$ the index of the last (least expensive) option in
            $tet_k$;
16:     return $CGG(f, B - e_{\min}, tet, optIdx)$;
17: $(b, optIdx) \leftarrow BSMCEM(f, tet)$;
18: return $CGG(f, b, tet, optIdx)$.
```

and expense $e_{k,n} = e_k(L_{k,n}, C_{k,n})$ as listed in Table III. The quadruples $\{(t_{k,n}, e_{k,n}, L_{k,n}, C_{k,n})\}$ in Table III are sorted in the ascending order of execution time, and are referred to as the time-expense table (TET) tet_k of job v_k. Here, each quadruple corresponds to an execution option of job v_k, so if its execution time and expense are both larger than those of another one, it would be deleted from tet_k;

In Alg. 3, BAWM first builds a time-expense table for each job by calling $bldTET()$, and then considers two special cases: i) If the input workflow is a pipeline, Alg. 1 is used to select an execution option for each job (Lines 5-7). Due to different execution speeds of a task running on VMs in different classes, Alg. 1 is merely a heuristic, not an approximation algorithm. ii) If the budget approaches the smallest sufficient budget $e_{\max}$ for the fastest execution of all jobs, BAWM keeps makespan the shortest and reduces the expense of jobs on non-critical paths by calling $save()$ to search for the minimal expense (Lines 8-12). In $save()$,

Algorithm 4: CGG()

```
Input: a workflow $f(G(V,A))$, remaining budget $b$, time-expense
       tables $\{tet_k\}$ and the option index $optIdx[k]$ of each job $v_k$
Output: makespan $t$ and remaining budget $b^*$
1: $t \leftarrow calMS(f, tet, optIdx)$;  $b^* \leftarrow b$;
2: while $b > 0$ do
3:     Compute the critical subgraph $g$ and makespan $t$ of workflow $f$
       based on the TETs $tet$ and option indices $optIdx$ of all jobs;
4:     if Remaining budget $b$ cannot be used to speed up any job in $g$ then
5:         break;
6:     $S \leftarrow \{v \in g |$ all critical paths traverse $v\}$;
7:     for all $v_k \in S$ do
8:         $i \leftarrow optIdx[k]$;
9:         for $n = i$-1 to 1 do
10:            $optIdx[k] \leftarrow n$; $t' \leftarrow calMS(f, tet, optIdx)$;
11:            $\Delta e(k,n) \leftarrow e_{k,n} - e_{k,i}$;
12:            if $\Delta e(k,n) < b$ then
13:                $ter(k,n) \leftarrow (t - t')/\Delta e(k,n)$;
14:        $optIdx[k] \leftarrow i$;
15:    Find job $k_1$ and option index $n_1$ with the maximum $ter(k_1, n_1)$;
16:    if $ter(k_1, n_1) > 0$ then
17:        $optIdx[k_1] \leftarrow n_1$;  $b \leftarrow b - \Delta e(k_1, n_1)$;  continue;
18:    $b^* \leftarrow b$;  $optIdx^* \leftarrow optIdx$;
19:    for all $v_k \in g - S$ do
20:        $n \leftarrow optIdx[k]$;
21:        $ter'(k, n-1) \leftarrow (t_{k,n} - t_{k,n-1})/(e_{k,n-1} - e_{k,n})$;
22:    Find job $k_2$ and option index $n_2$ with the maximum $ter'(k_2, n_2)$;
23:    if $ter'(k_2, n_2) > 0$ then
24:        $optIdx[k_2] \leftarrow n_2$;  $b \leftarrow b - \Delta e(k_2, n_2)$;
25: return $(t, b^*)$.
```

we calculate the ratio of the expense decrease over the time increase for each slower execution option of each non-critical job, and then downgrade the job with the highest ratio to the corresponding execution option one at a time.

As shown in Alg. 3, the key idea of BAWM is as follows. Each workflow mapping consists of two components: critical-subgraph-greedy (CGG) and binary search based on makespan-constrained expense minimization (BSMCEM). A CP is the longest execution path in a workflow, which can be calculated in linear time. Here, we define the union of all critical paths as a critical subgraph, which can also be calculated in linear time. CGG is designed to find a local optimal solution; BSMCEM is designed to find a feasible solution near a global optimal solution with some remaining budget. Therefore, if the budget approaches the smallest allowable budget $e_{\min}$ for a feasible workflow mapping scheme, BAWM uses $CGG()$ in Alg. 4 to search for the shortest makespan (Lines 13-16). Otherwise, BAWM first uses $BSMCEM()$ in Alg. 5 to find a near-optimal feasible solution, and then starts $CGG()$ with this solution and utilizes its remaining budget to achieve a better solution (Lines 17-18).

B. Critical-subgraph Greedy

In Alg. 4, we allocate the remaining budget based on the currently selected option for each job to accelerate the workflow execution. We select those jobs all CPs traverse as critical points and upgrade their options to faster ones, with an attempt to maximize the impact of a local speed-up on the global end-to-end delay. For each such job, we first calculate its local maximum ratio of the makespan decrease over the expense increase and record the corresponding option index,

Algorithm 5: BSMCEM()

```
Input: A budget-constrained workflow $f(G(V,A),B)$ and TETs $tet$
Output: remaining budget $b$ and the option indices $optIdx$ of all jobs
 1: Find the minimum makespan as $t_{\min}$ when each job selects the fastest
    execution option and the maximum makespan as $t_{\max}$ when each job
    selects the least expensive execution option;
 2: $t_F \leftarrow t_{\min}/(1+\epsilon_2)$;  $t_S \leftarrow t_{\max}$;
 3: $t' \leftarrow$ -1;  // $t'$ is the time constraint of the latest successful mapping
 4: while $t_S - t_F > \Delta t$ do
 5:     $t \leftarrow (t_S + t_F)/2$;    $(t^C, e, optIdx) \leftarrow MAWMEM(t, f, tet)$;
 6:     if $e > B$ then
 7:         if $t_F < t^C < t$ then
 8:             $t_F \leftarrow t^C$;
 9:         else
10:             $t_F \leftarrow t$;
11:     else
12:         if $t_S > t^C$ then
13:             $t_S \leftarrow t^C$; $t' \leftarrow t$;
14:         else
15:             break;
16:     Cancel the mapping of all the jobs in $V$;
17: if $t' \geq 0$ then
18:     $(t^C, e, optIdx) \leftarrow MAWMEM(t', f, tet)$;
19:     $b \leftarrow B - save(f, tet, optIdx)$;
20: else
21:     $b \leftarrow B - e_{\min}$;
22: return $(b, optIdx)$.
```

Algorithm 6: MAWMEM()

```
Input: makespan constraint $T$, workflow $f(G(V,A))$, and TETs $tet$
Output: makespan $t^C$, expense $e$, the option indices $optIdx$ of all jobs
 1: $t_k^{LF} \leftarrow +\infty$ for $\forall v_k \in f$;    $t_K^{LF} \leftarrow T$ for the end job $v_K$ in $f$;
 2: $G' \leftarrow G$;    // $G'$ is a subgraph of all the unmapped jobs in $G$.
 3: while $G' \neq \emptyset$ do
 4:     Find the critical path $cp$ ending at a job with the earliest LFT in
        $G'$ according to $\{t_{i,k}(L'_k) | e_{i,k}(L'_k) = \min_{C_{i'} \subset Y} e_{i',k}(L'_k)\}$;
 5:     if $PM(cp, tet) =$ False then
 6:         $v_k \leftarrow$ the last mapped job in $cp$;
 7:         $D(v_k) \leftarrow \{$the downstream jobs of $v_k$ in $G\}$;
 8:         for all $v_{k'} \in D(v_k)$ s.t. $t_{k'}^S < t_k^F$ do
 9:             Cancel the mapping of $v_{k'}$, and add it and its associated
                precedence constraints back to $G'$;
10:     $G' \leftarrow G' - \{v \in cp | v$ is mapped$\}$;
11: $t^C \leftarrow calMS(f, tet, optIdx)$;    $e \leftarrow calExp(f, tet, optIdx)$;
12: return $(t^C, e, optIdx)$.
```

and then select the job that achieves the global maximum ratio over the entire critical subgraph for rescheduling. If none of the jobs at the critical points can be accelerated, we upgrade one of the rest in the critical subgraph by one level with the same selection method as above.

C. Binary Search based on Makespan-constrained Expense Minimization

The CP plays a significant role in the performance optimization of workflow mapping. However, it is difficult to balance the expenses between the jobs on the CP and the jobs on noncritical paths (NCPs) as the CP might change during the workflow mapping process. Towards this end, we first design an heuristic to solve another optimization problem, MAkespan-constrained Workflow Mapping for Expense Minimization (MAWMEM), which has the same decision version as BCWM4C. The difference between BCWM4C and MAWMEM is that their objective and one of the constraints

Algorithm 7: PM()

```
Input: a pipeline $pl$ with its EST $pl.est$ and LFT $pl.lft$ and TETs $tet$
Output: a boolean variable to indicate whether mapped jobs in $pl$
        follow precedence constraints
 1: Label the index $k$ of each job in $pl$ from 1 to the length of $pl$;
 2: Update TEST $t'_{ES}(v_k)$ and TLFT $t'_{LF}(v_k)$ for $\forall v_k \in pl$;
 3: if $\sum_{v_k \in pl} t_{k,1} > pl.lft - pl.est$ then
 4:     $t_1^S \leftarrow pl.est$;
 5:     for $v_k \in pl$ do
 6:         if $k > 1$ then
 7:             $t_k^S \leftarrow \max\{t_{k-1}^F, t'_{ES}(v_k)\}$;
 8:         $t_k^F \leftarrow t_k^S + t_{k,1}$;  $L_k \leftarrow L_{k,1}$;
 9:         if $t_k^F > t'_{LF}(v_k)$ then
10:             return False.
11:     return False.
12: Use Alg. 2 to calculate the execution option index $optIdx[k]$, AST $t_k^S$
    and AFT $t_k^F$ for each job $v_k \in pl$ based on $\{tet_k | v_k \in pl\}$;
13: for $v_{k+1} \in pl$ do
14:     if $t_k^F > t'_{LF}(v_k)$ or $t_k^F < t'_{ES}(v_{k+1})$ then
15:         Let $pl(1,k)$ be the sub-pipeline of $pl$ from its 1st to the $k$-th job;
16:         $pl(1,k).est \leftarrow pl.est$;
17:         if $t_k^F > t'_{LF}(v_k)$ then
18:             $pl(1,k).lft \leftarrow t'_{LF}(v_k)$;
19:         else
20:             $pl(1,k).lft \leftarrow \min\{t'_{ES}(v_{k+1}), t'_{LF}(v_k)\}$
21:         Clear $optIdx[k]$, $t_k^S$ and $t_k^F$ for each job $v_k$ in $pl$;
22:         return $PM(pl(1,k), tet)$;
23: Map $v_k$ from $t_k^S$ to $t_k^F$ according to the $optIdx[k]$-th option in $tet_k$;
24: return True.
```

are swapped. The objective of MAWMEM is

$$\min_{\mathfrak{M}} \sum\nolimits_{v_k \in V} e_k,$$

and the first constraint of MAWMEM is

$$t^C \leq T.$$

Accordingly, the framework of BSMCEM in Alg. 5 is as follows. Initially, we set the infeasible makespan t_F to be the minimum makespan $t_{\min}$ minus a tiny amount and the feasible makespan t_S to be the maximum makespan $t_{\max}$, and seek for workflow mapping with a minimal makespan by binary search (Lines 4-16). Then, BSMCEM searches for the maximal remaining budget with this minimal makespan fixed by calling $save()$.

For the sake of clarification, we define the following notations. If all the preceding jobs of job v_k are mapped, its earliest start time (EST) t_k^{ES} is the maximum AFT of its preceding jobs; if all the succeeding jobs of job v_k are mapped, its last finish time (LFT) t_k^{LF} is the minimum AST of its succeeding jobs. The EST of the start job is 0, and the LFT of the end job is a makespan bound. If there exist unmapped preceding and succeeding jobs of v_k, its temporary earliest start time (TEST) $t'_{ES}(v_k)$ and temporary last finish time (TLFT) $t'_{LF}(v_k)$ can be calculated based on only its mapped preceding and succeeding jobs, respectively. The EST and LFT of a pipeline are the EST of its first job and LFT of its end job, respectively.

We consider MAWMEM and design a heuristic algorithm in Alg. 6, whose key idea is as follows. Each workflow mapping consists of two components: iterative CP selection and pipeline mapping. The algorithm starts with computing an initial CP according to the execution time of each job with a maximum number of tasks running in parallel in the least expensive

VM class, followed by a pipeline mapping process. Then, it iteratively computes a CP with the earliest last finish time (LFT) from the remaining unmapped workflow branches based on the same execution time of a job as above and performs a pipeline mapping of the computed CP until there are no branches left. If the mapping of the last mapped job on the CP violates the precedence constraint with its downstream jobs, all these downstream jobs in violation would be unmapped. Note that the first job on each previous mapped CP would not be cancelled because the CP with the earliest LFT is selected and mapped in each iteration.

In pipeline mapping in Alg. 7, due to the homogeneity of tasks in a job, we map all the tasks in the same job onto VM instances in a single class, hence using Alg. 2 to balance the trade-off between execution time and expense for each job on a pipeline (Line 12). When the jobs in a pipeline are mapped in their execution order, we check if everyone respects precedence constraints, and remap a pipeline with the updated LFT and length, if needed (Lines 13-22).

Since CGG is of $O(I^2K^2L'^2|A|)$ and BSMCEM is of $O(IK^5L'\log(t_{\max}/\Delta t)/\epsilon_2)$, the time complexity of BAWM is $O(I^2K^2L'^2|A| + IK^5L'\log(t_{\max}/\Delta t)/\epsilon_2 + I^2K^2L'^2/\epsilon_1)$, where I is the number of classes of VM types, K is the number of jobs, L' is the maximum possible number of tasks in a job.

VI. Performance Evaluation

A. Simulation Settings

We conduct simulations to evaluate our mapping model, referred to as Job-VMC, which maps each job with adaptive task partitioning onto a set of VM instances of types in the same class, in comparison with two existing mapping models: one mapping each task onto a VM instance, referred to as Task-VMI [17], and the other mapping each job with a preset number of tasks onto a set of VM instances of the same VM type, referred to as Job-VMT [18]. Then, we conduct simulations to evaluate the performance and financial cost of BAWM in our mapping model in comparison with existing algorithms: global-greedy-budget (GGB) in [17], gradual refinement (GR) in [17], and critical-path-greedy (CPG) in [18].

We generate a series of random workflows, in each of which the number of precedence constraints is set to 1.5 times the number of jobs, if possible. The workload of a job is randomly selected between 0.06×10^{15} and 2.16×10^{15} CPU cycles when running in serial. The workload $w(l)$ of a job with $l > 1$ tasks is randomly selected between $w(l-1)[1 + 0.3/(l-1)]$ and $w(l-1)[1 + 0.7/(l-1)]$. The percentage of time executing CPU-burst instructions of each job on VMs in each class is randomly selected between 0.1 and 1. The memory demand of a task in each job is randomly selected from 0.5GB to 3.5GB at an interval of 0.5GB.

We conduct the simulation on five classes of VM types for different usage purposes, consisting of 20 VM types as tabulated in Table IV, based on real-life VM types in Amazon EC2. The parameters ϵ_1 in Alg. 1 and ϵ_2 in Alg. 2 are set to 0.05 and 0.01, respectively. By default, each data point denotes the average of 1000 runs with standard deviations; the workflow size and the number of VM types are set to be 100 jobs and 20, respectively; the maximum number L' of tasks for each job is randomly selected between 30 and 50; the budget factor β, which is a factor determining the actual budget B based on a budget range $[e_{\min}, e_{\max}]$ as $B = e_{\min} + \beta(e_{\max} - e_{\min})$, is set to 0.3.

In each simulation, the mapping success rate (MSR) is defined as the ratio of the number of workflow requests (i.e. pairs of a workflow instance and a budget), each of which has a feasible mapping scheme, to the total number of workflow requests in a mapping model. The performance improvement, i.e. makespan reduction, over other algorithms in comparison is defined as

$$Imp(Other) = \frac{MS_{Other} - MS_{BAWM}}{MS_{Other}} \cdot 100\%,$$

where MS_{Other} is the makespan achieved by the other algorithm, and MS_{BAWM} is the makespan achieved by BAWM. The financial improvement of BAWM over another algorithm in comparison is defined as the remaining budget percentage of BAWM minus that of another algorithm.

B. Mapping Success Rate for Mapping Models

We first examine the performance of Job-VMC, Job-VMT, and Task-VMI in terms of MSR with various problem sizes under different budget constraints. We consider 20 different problem sizes from small to large scales, indexed from 1 to 20, as tabulated in Table V. Each problem size is defined as a

TABLE IV
SPECIFICATIONS FOR VIRTUAL MACHINE TYPES

VM Type	Class Name	Processor Speed (GHz)	# of cores	Memory (MB)	Price ($/hour)
1	General Purpose	3.25	1	3840	0.067
2	General Purpose	3.25	2	7680	0.134
3	General Purpose	3.25	4	15360	0.268
4	General Purpose	3.25	8	30720	0.536
5	Compute Optimized	3.5	2	3840	0.105
6	Compute Optimized	3.5	4	7680	0.21
7	Compute Optimized	3.5	8	15360	0.42
8	Compute Optimized	3.5	16	30720	0.84
9	Compute Optimized	3.5	32	61440	1.68
10	Memory Optimized	3.25	2	15360	0.166
11	Memory Optimized	3.25	4	31232	0.332
12	Memory Optimized	3.25	8	62464	0.664
13	Memory Optimized	3.25	16	124928	1.328
14	Memory Optimized	3.25	32	249856	2.656
15	Storage Optimized	3.5	4	31232	0.853
16	Storage Optimized	3.5	8	62464	1.706
17	Storage Optimized	3.5	16	124928	3.412
18	Storage Optimized	3.5	32	249856	6.824
19	GPU Instances	3.25	8	15360	0.65
20	GPU Instances	3.25	32	61440	2.6

TABLE V
PROBLEM SIZES

Index	$(\|V\|, L', \|Y\|)$	Index	$(\|V\|, L', \|Y\|)$
1	(5, 15-26, 1)	11	(55, 23-38, 11)
2	(10, 16-27, 2)	12	(60, 24-40, 12)
3	(15, 17-28, 3)	13	(65, 24-41, 13)
4	(20, 18-30, 4)	14	(70, 25-42, 14)
5	(25, 18-31, 5)	15	(75, 26-43, 15)
6	(30, 19-32, 6)	16	(80, 27-45, 16)
7	(35, 20-33, 7)	17	(85, 27-46, 17)
8	(40, 21-35, 8)	18	(90, 28-47, 18)
9	(45, 21-36, 9)	19	(95, 29-48, 19)
10	(50, 22-37, 10)	20	(100, 30-50, 20)

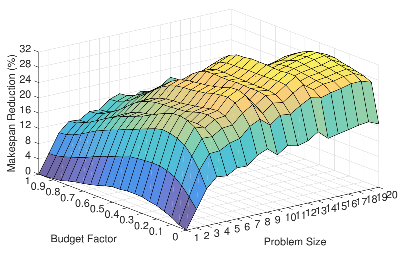


Fig. 3. The average performance improvement of BAWM over GGB in 400 instances for each budget factor and each problem size.

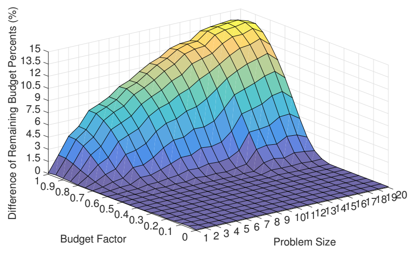


Fig. 4. The average financial improvement of BAWM over GGB in 400 instances for each budget factor and each problem size.

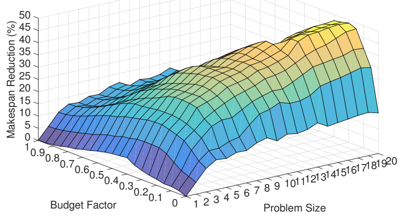


Fig. 5. The average performance improvement of BAWM over GR in 400 instances for each budget factor and each problem size.

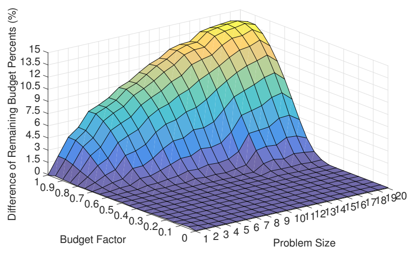


Fig. 6. The average financial improvement of BAWM over GR in 400 instances for each budget factor and each problem size.

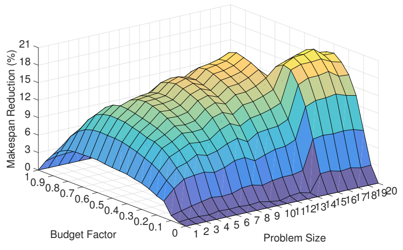


Fig. 7. The average performance improvement of BAWM over CPG in 400 instances for each budget factor and each problem size.

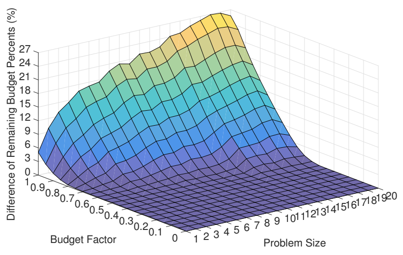


Fig. 8. The average financial improvement of BAWM over CPG in 400 instances for each budget factor and each problem size.

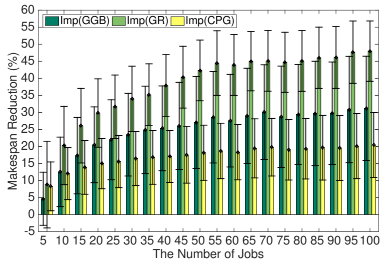


Fig. 9. The performance improvement of BAWM in 1000 instances for each workflow size.

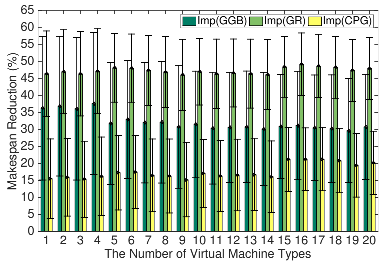


Fig. 10. The performance improvement of BAWM in 1000 instances for each number of VM types.

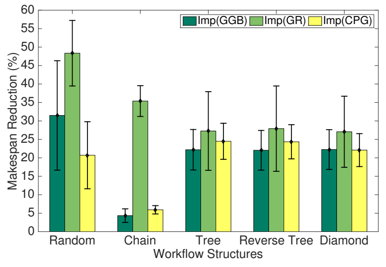
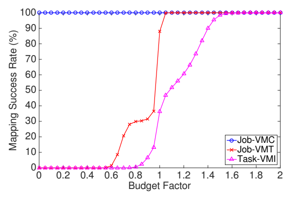
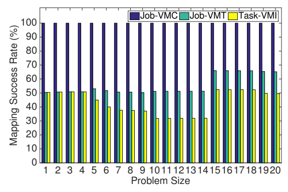


Fig. 11. The performance improvement of BAWM in 1000 instances for each workflow structure.



(a)



(b)

Fig. 2. Mapping success rate: (a) in 2,000,000 instances (20 different problem sizes $\times$ 100,000 random workflow instances) for each budget factor; (b) in 2,100,000 instances (21 different budget factors $\times$ 100,000 random workflow instances) for each problem size.

triple of the number $|V|$ of jobs per workflow, the maximum number L' of tasks per job, and the number $|Y|$ of VM types. The budget factor on the budget range $[e_{\min}, e_{\max}]$ calculated based on the Job-VMC mapping model varies from 0 to 2 in the evaluation. The MSR of these mapping models with different budget factors and problem sizes are plotted in Fig. 2(a) and Fig. 2(b), respectively. Fig. 2(a) shows that Job-VMC achieves 100% MSR when $\beta \geq 0$, while Job-VMT and Task-VMI do not achieve 100% MSR until β reaches 1.1 and 1.7, respectively. Fig. 2(b) shows that with different problem sizes, the MSR of BAWM is 100%, while the MSRs of Job-VMT and Task-VMI are only between 50.3% and 66.0%, and between 31.8% and 52.4%, respectively.

C. *Makespan for Mapping Algorithms*

1) *Problem Size and Budget Scale*

We evaluate the performance of GGB, GR, CPG, and BAWM in the Job-VMC mapping model in terms of makespan and remaining budget with problem sizes from index 1 to 20 under budget constraints from 0 to 1 at an interval of 0.05. We plot the average makespan reduction and financial improvement of BAWM over GGB, GR, and CPG with different budget factors and problem sizes in Figs. 3-8, respectively. These measurements show that BAWM reduces makespan by up to 30.8%, 49.1%, and 20.8%, and by 20.5%, 24.3%, and

8.4% on average in comparison with GGB, GR and CPG, respectively. The performance optimization of GGB and GR is limited by the bounds of each stage, and thus is inferior to that of BAWM. CPG, recursively improving the performance of the CP, ignores the impact of jobs in the non-critical paths on the budget usage, and thus causes unnecessary financial waste and limits its performance improvement. When the budget reaches $e_{\min}$ (or $e_{\max}$), all the jobs run the slowest (or fastest), and CPG and BAWM obtain the optimal makespan in polynomial time, so there is no performance improvement of BAWM over CPG in both scenarios. Furthermore, when BAWM performs other three algorithms in terms of makespan, it increases the percentage of remaining budget by up to 14.3%, 14.3%, and 26.9%, and by 2.9%, 2.9%, and 4.3% on average in comparison with GGB, GR and CPG, respectively. With tight budgets, all these algorithms exhaust budgets to seek for the least end-to-end delay; with loose budgets, since all the jobs do not need to run the fastest to achieve the minimum makespan, BAWM save a significant amount of financial cost.

2) *Workflow Size*

We execute GGB, GR, CPG, and BAWM in the Job-VMC model under different workflow sizes for scalability evaluation. We plot the makespan reduction in Fig. 9, where we observe that as the number of jobs increases, BAWM improves the performance by 4.7% to 31.2%, 8.8% to 48.0%, and 8.3% to 20.4% in comparison with GGB, GR, and CPG, respectively, hence exhibiting a satisfactory scalability property with respect to the workflow size.

3) *The Number of Virtual Machine Types*

We also run these algorithms in the Job-VMC model under different numbers of VM types. The set of available VM types are selected in the order of VM types listed in Table IV every time. The makespan reduction of BAWM over GGB, GR, and CPG is plotted in Fig. 10, which shows that BAWM improves the performance by 29.6% to 37.6%, 46.0% to 49.2%, and 15.2% to 21.2% in comparison to GGB, GR, and CPG, respectively.

4) *Workflow Structure*

We evaluate the performance of these algorithms in the Job-VMC model with different workflow structures, including a random shape, a chain, a tree, a reverse tree, and a diamond. The makespan reduction is plotted in Fig. 11, which shows that BAWM improves the performance by 31.5%, 4.3%, 22.2%, 22.1% and 22.2%, by 48.4%, 35.4%, 27.3%, 27.9% and 27.1%, and by 20.7%, 5.9%, 24.5%, 24.4% and 22.1% in comparison with GGB, GR, and CPG for five different workflow structures, respectively. Although there is less room for performance improvement in such a simple workflow structure as pipeline, BAWM still achieves considerable performance improvement as a result of incorporating Alg. 1 for the first special case.

VII. Conclusion

We investigated the properties of moldable jobs and designed a big-data workflow mapping model in clouds, based on which, we formulated a strongly NP-complete workflow mapping problem to minimize workflow makespan under a budget constraint. We designed an FPTAS for a special case with a pipeline on VMs in a single class and a heuristic for a generalized problem with an arbitrary workflow on VMs in multiple classes. The performance superiority of the proposed solution was illustrated by extensive simulation-based results in comparison with existing mapping models and algorithms. We plan to implement and evaluate the proposed mapping solution using real-life scientific workflows in public clouds.

Acknowledgment

This research is sponsored by U.S. Department of Energy's Office of Science under Grant No. DE-SC0015892 and National Science Foundation under Grant No. CNS-1560698 with New Jersey Institute of Technology, and National Nature Science Foundation of China under Grant No. 61472320 with Northwest University, P.R. China.

References

[1] H. Arabnejad and J. G. Barbosa. A budget constrained scheduling algorithm for workflow applications. *Springer J. Grid Comp.*, 12(4):665–679, 2014.

[2] C.-Y. Chen and C.-P. Chu. A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE TPDS*, 24(8):1479–1488, 2013.

[3] S. Chen, M. Song, and S. Sahni. Two techniques for fast computation of constrained shortest paths. *IEEE/ACM Tran. on Net.*, 16(1):105–115, 2008.

[4] M. Drozdowski. *Scheduling for Parallel Processing*. Springer-Verlag London, 2009.

[5] J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Disc. Math.*, 2(4):473–487, 1989.

[6] F. Ergun, R. Sinha, and L. Zhang. An improved FPTAS for restricted shortest path. *Info. Processing Letters*, 83(5):287–291, 2002.

[7] T. F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

[8] B. Huang, S. Babu, and J. Yang. Cumulon: Optimizing statistical data analysis in the cloud. In *Proc. of ACM SIGMOD*, pages 1–12, New York, NY, USA, Jun 2013.

[9] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. *J. of Computer and System Sci.*, 78(1):245–259, 2012.

[10] Q. Jiang, Y. C. Lee, and A. Y. Zomaya. Executing large scale scientific workflow ensembles in public clouds. In *Proc. of IEEE ICPP*, pages 520–529, Beijing, China, Sep 2015.

[11] K. Makarychev and D. Panigrahi. Precedence-constrained scheduling of malleable jobs with preemption. In *Proc. of ICALP*, pages 823–834, Copenhagen, Denmark, Jul 2014.

[12] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proc. of ACM/IEEE SC*, pages 1–12, Seatle, WA, USA, Nov 2011.

[13] V. Nagarajan, J. Wolf, A. Balmin, and K. Hildrum. FlowFlex: Malleable scheduling for flows of MapReduce jobs. In *Proc. of ACM/IFIP/USENIX Middleware*, pages 103–122, Beijing, China, Dec 2013.

[14] M. A. Rodriguez and R. Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Trans. on Cloud Comp.*, 2(2):222–235, 2014.

[15] T. Sandholm and K. Lai. Dynamic proportional share scheduling in hadoop. In *Proc. of Int. Work. on JSSPP*, pages 110–131, Atlanta, GA, USA, Apr 2010.

[16] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag Berlin Heidelberg, 2003.

[17] Y. Wang and W. Shi. Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds. *IEEE Tran. on Cloud Comp.*, 3(2):306–319, 2015.

[18] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *IEEE Tran. on Cloud Comp.*, 3(2):169–181, 2015.

[19] W. Zheng and R. Sakellariou. Budget-deadline constrained workflow planning for admission control. *Springer J. Grid Comp.*, 11(4):633–651, 2013.