Energy-efficient Mapping of Large-scale Workflows Under Deadline Constraints in Big Data Computing Systems

Tong Shu and Chase Q. Wu

Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA

Abstract

Large-scale workflows for big data analytics have become a main consumer of energy in data centers where moldable parallel computing models such as MapReduce are widely applied to meet high computational demands with time-varying computing resources. The granularity of task partitioning in each moldable job of such big data workflows has a significant impact on energy efficiency, which remains largely unexplored. In this paper, we analyze the properties of moldable jobs and formulate a workflow mapping problem to minimize the dynamic energy consumption of a given workflow request under a deadline constraint in big data systems. Since this problem is strongly NP-hard, we design a fully polynomial-time approximation scheme (FPTAS) for a special case with a pipeline-structured workflow on a homogeneous cluster and a heuristic for the generalized problem with an arbitrary workflow on a heterogeneous cluster. The performance superiority of the proposed solution in terms of dynamic energy saving and deadline missing rate is illustrated by extensive simulation results in comparison with existing algorithms, and further validated by real-life workflow implementation and experimental results in Hadoop/YARN systems.

Keywords: Big data, workflow mapping, green computing

Introduction

Next-generation applications in science, industry, and business domains are producing colossal amounts of data, now frequently termed as "big data", which must be analyzed in a timely manner for knowledge discovery and technological innovation. Among many practical computing solutions,

*Some preliminary results in this manuscript were published in WORKS'16 in conjunction with SC'16 [1].

*Please send all correspondence to Wu.

Email address: {ts372, chase.wu}@njit.edu (Tong Shu and Chase Q. Wu)

workflows have been increasingly employed as an important technique for big data analytics, and consequently such big data workflows have become a main consumer of energy in data centers. Most existing efforts on green computing were focused on independent MapReduce jobs and traditional workflows comprised of serial/rigid programs. Energy efficiency of large-scale workflows in big data systems such as Hadoop still remains largely unexplored.

Modern computing systems achieve energy saving mainly through two types of techniques, i.e. task consolidation to reduce static energy consumption (SEC) by turning off idle servers, and load balancing to reduce dynamic energy consumption (DEC) through dynamic voltage and frequency scaling (DVFS), or a combination of both. However, these techniques are not sufficient to address the energy efficiency issue of big data workflows because i) frequently switching on and off a server may reduce its lifespan or cause unnecessary peaks of power consumption, and ii) DVFS may not be always available on all servers in a cluster. Therefore, we direct our efforts to workflow mapping for dynamic energy saving by adaptively determining the degree of parallelism in each MapReduce job to mitigate the workload overhead while meeting a given performance requirement.

Parallel jobs are generally categorized into three classes with flexibility from low to high: rigid jobs exemplified by multi-threaded programs running on a fixed number of processors, moldable jobs exemplified by MapReduce programs running on any number of processors decided prior to execution, and malleable jobs running on a variable number of processors at runtime [2]. A moldable job typically follows a performance model where the workload of each component task decreases and the total workload, proportional to DEC, increases as the number of allotted processors increases [3]. The validity of this model has been verified by many real-life parallel programs in various big data domains and will serve as a base of our workflow mapping solution for energy saving of big data workflows.

In this paper, we construct analytical cost models and formulate a workflow mapping problem to minimize the DEC of a workflow under deadline and resource constraints in a Hadoop cluster. This problem is strongly NP-hard because a subproblem to minimize the makespan of independent jobs on identical machines under a single resource constraint without considering energy cost has been proved to be strongly NP-hard [4]. In our problem, it is challenging to balance the trade-off between energy cost and execution time of each component job to determine their respective completion time in MapReduce workflows, regardless of several previous efforts in traditional workflows, such as the partial critical path and minimum dependency methods in [5, 6].

We start with a special case with a pipeline-structured workflow (a set of linearly arranged jobs with a dependency between any two neighbors along the line) on a homogeneous cluster. We prove this special case to be weakly NP-complete and design a fully polynomial-time approximation scheme (FPTAS) of time complexity linear with respect to $1/\epsilon$. By leveraging the near optimality and low time complexity of our FPTAS, we design a heuristic for the generalized problem with a directed acyclic graph (DAG)-structured workflow on a heterogeneous cluster. This heuristic iteratively selects the longest chain of unmapped jobs from the workflow and applies our FPTAS to the selected pipeline while taking machine heterogeneity into consideration.

In sum, our work makes the following contributions to the field.

- Our work validates with experimental results that the DEC of a moldable job increases with the number of parallel tasks, and to the best of our knowledge, is among the first to study energy-efficient mapping of big data workflows comprised of moldable jobs in Hadoop systems.
- We prove a deadline-constrained pipeline-structured workflow mapping problem for minimum total (energy) cost to be weakly NP-complete and design an FPTAS, whose performance is illustrated through real-life workflow implementation and extensive experimental results using the Oozie workflow engine in Hadoop/YARN systems.
- The performance superiority of the proposed heuristic for the general workflow mapping problem in terms of dynamic energy saving and deadline missing rate is illustrated by extensive simulation results in Hadoop/YARN in comparison with existing algorithms.

The rest of the paper is organized as follows. Section 2 provides a survey of related work. Section 3 formulates a MapReduce workflow mapping problem. We prove a special case to be weakly NP-complete and design an FPTAS for it in Section 4, and design a heuristic for the generalized problem in Section 5. Section 6 evaluates the performance and Section 7 concludes our work.

Related Work

Energy efficiency in Hadoop Systems

Energy-efficient Data Placement

A large number of research efforts have been made to optimize the data replication scheme in Hadoop distributed file system (HDFS) so that data nodes can be turned off without affecting data availability. To allow scale-down of an operational Hadoop cluster, Leverich *et al.* introduced

the notion of a covering subset (CovSet) for HDFS, a small subset of machines, within which one replica of every block is stored [7]. Lang *et al.* proposed the all-in strategy (AIS) that turns off all servers for energy saving and turns on all servers to accommodate all tasks as fast as possible when the task queue is large enough. They demonstrated the superiority of AIS compared to CovSet in terms of response time and energy cost when the transition time of nodes to and from a low power state is relatively small compared with the total workload execution time [8]. Amur *et al.* proposed to maintain the primary replica of each data block on the primary nodes that are always active and store B/n secondary replicas on the n-th node on the expansion-chain (B is the total number of replicas), which denotes the order in which nodes must be turned on/off to scale performance up/down to support the equal-work layout for power-proportionality [9]. Chen *et al.* developed BEEMR, an energy-efficient MapReduce workload manager motivated by an empirical analysis of real-life traces of MapReduce workloads from Facebook [10]. The key insight is that interactive jobs often operate on a small fraction of data, and thus can be served by a small pool of dedicated machines, while jobs that are less time sensitive can run in a batch manner on the rest of the cluster. Energy savings come from aggregating the execution of less time-sensitive jobs in the batch zone to achieve high utilization, and transitioning idle machines in the batch zone to a low-power state. These techniques showed dramatic improvements in energy saving at the file system level. Our research on job scheduling is orthogonal to these efforts, and hence adds an additional level of energy efficiency to Hadoop systems.

Energy-efficient MapReduce Job Scheduling

Dynamic Voltage Frequency Scaling (DVFS): The DVFS technology has been widely adopted for energy saving in computing systems. Bampis *et al.* focused on the minimization of the total weighted completion time for a set of MapReduce jobs under a given energy constraint, and used a linear programming relaxation method to derive a polynomial-time constant-factor approximation algorithm [11].

Heterogeneous Computing Environments: Since servers in large-scale clusters are typically upgraded or replaced in an incremental manner, many techniques consider hardware heterogeneity of Hadoop clusters for energy saving. Cardosa *et al.* proposed static virtual machine (VM) placement algorithms to minimize the cumulative machine uptime of all physical machines (PMs), based on two principles: spatial fitting of VMs on PMs to achieve high resource utilization according to complementary resource requirements from VMs, and temporal fitting of PMs with VMs having

similar runtime to ensure that a server runs at a high utilization level throughout its uptime [12]. Mashayekhy *et al.* modeled the energy-aware static task scheduling of a MapReduce job as an Integer Programming problem, and designed two heuristics that assign map/reduce tasks to machine slots to minimize energy consumption while satisfying the service level agreement (SLA) [13]. Cheng *et al.* proposed a heterogeneity-aware dynamic task assignment approach using ant colony optimization, referred to as E-Ant, to minimize the overall energy consumption of MapReduce applications with heterogeneous workloads in a heterogeneous Hadoop cluster without *a priori* knowledge of workload properties [14]. The use of the ant colony algorithm in the Hadoop scheduler is based on an assumption that there exist a large number of homogeneous tasks in a MapReduce job. However, an excessively large number of tasks in a parallel job may incur very high overhead (compared with the payload itself), hence leading to a significant waste of energy and delaying the job completion time.

Renewable Energy: Several efforts were focused on utilizing renewable energy in the operation of Hadoop clusters. Goiri *et al.* proposed a framework, GreenHadoop, for a data center powered by renewable (green) energy and by carbon-intensive (brown) energy from the electrical grid as a backup. It dynamically schedules MapReduce jobs to minimize brown energy consumption by delaying background computations within their time bounds to match the green energy supply that is not always available [15]. Cheng *et al.* designed a scheduler for a Hadoop cluster powered by mixed brown and green energy, which dynamically determines resource allocation to heterogeneous jobs based on the estimation of job completion time and the prediction of future resource availability [16]. Despite the salient features for energy cost saving enabled by mixed energy supplies, at present there is no mature technology to support seamless switch between green and brown energy supplies or bring down the cost for storing renewable energy in such MapReduce frameworks.

Overhead Reduction: A few efforts were devoted to workload overhead reduction for energy saving. Sharma *et al.* designed a dynamic scheduler for interactive and batch MapReduce jobs in hybrid physical and virtual environments to boost resource utilization and energy saving through workload consolidation based on virtualization and avoid virtualization-incurred overhead by executing "heavy" jobs immediately on PMs [17]. The majority of existing efforts targeted the first generation of Hadoop. The work on the second generation of Hadoop, i.e. YARN, is still quite limited. Li *et al.* proposed a suspend-resume mechanism in YARN to mitigate the overhead of preemption in cluster scheduling, and used a check pointing mechanism to save the states of jobs

for resumption [18]. Their approach dynamically selects appropriate preemption mechanisms based on the progress of a task and its suspend-resume overhead to improve job response time and reduce energy consumption. As opposed to preemptive scheduling of interactive applications in their work, our work is focused on energy saving in non-preemptive scheduling of MapReduce jobs in the background.

Energy-efficient Workflow Scheduling

Many efforts were made on energy-efficient scheduling of workflows comprised of precedence-constrained serial programs. Some of these approaches targeted virtualized environments [19] by migrating active VMs onto energy-efficient PMs in time [20] or consolidating applications with complementary resource requirements [21]. Zhu *et al.* developed a workflow scheduling framework, pSciMapper, which consists of two major components: i) online power-aware consolidation, based on available information on the utilization of CPU, memory, disk, and network by each job, and ii) offline analysis including a hidden Markov model for estimating resource usage per job and kernel canonical correlation analysis for modeling the resource-time and resource-power relationships [21].

Other approaches were focused on physical clusters as follows. Lee *et al.* proposed a static workflow schedule compaction algorithm to consolidate the resource use of a workflow schedule generated by any scheduling algorithm in homogeneous environments [22], and designed two static energy-conscious workflow scheduling algorithms based on DVFS in heterogeneous distributed systems [23]. In [24], three types of DVFS-based heuristics, namely, prepower-determination, postpower-determination, and hybrid algorithms, were designed to solve a static problem of joint power allocation and workflow scheduling for schedule length (or energy consumption) minimization under an energy constraint (or a time constraint). Zhang *et al.* proposed a DVFS-based heuristic to statically maximize workflow reliability under a energy constraint in a heterogeneous cluster [25], and designed a Pareto-based bi-objective genetic algorithm to achieve low energy consumption and high system reliability for static workflow scheduling [26]. Zotkiewicz *et al.* proposed a communication-aware minimum-dependency energy-efficient DAG (MinD+ED) scheduling strategy for SaaS applications in heterogeneous data centers, which statically determines virtual deadlines of individual tasks by favoring tasks less dependent on others and then dynamically assigns tasks based on the load of network links and servers [6]. The above work only considers serial or rigid jobs in workflows, while our work is focused on moldable jobs in big data computing systems.

Malleable Job Scheduling

Some efforts have been made to minimize the completion time of a workflow comprised of malleable jobs [27, 28, 29], but there exist relatively limited efforts on moldable/malleable job scheduling for energy efficiency. Sanders *et al.* designed a polynomial-time optimal solution and an FPTAS to statically schedule independent malleable jobs with a common deadline for energy consumption minimization based on the theoretical power models of a single processor using the DVFS technology, i.e. $p = f^{\alpha}$ and $p = f^{\alpha} + \delta$, respectively, where f is CPU frequency and δ is the constant static power consumption [30]. Different from these theoretical models, our work employs measurement-based power consumption models and performs workflow mapping to reduce the computing overhead and thus improve the energy efficiency of big data workflows. To the best of our knowledge, our work is among the first to study energy-efficient mapping of big data workflows comprised of moldable jobs in Hadoop systems.

Problem Formulation

Cost Models

Cluster Model

We consider a heterogeneous Hadoop cluster consisting of a set M of machines connected via high-speed switches, which can be partitioned into homogeneous sub-clusters $\{C_l\}$. Each machine m_i is equipped with N_i homogeneous CPU cores of speed p_i and a shared memory of size o_i. For the entire cluster, a central scheduler maintains an available resource-time (ART) table R, which records the number $N_i^A(t) \leq N_i$ of idle CPU cores and the size $o_i^A(t) \leq o_i$ of available memory in each machine m_i at time t.

Workflow Model

We consider a user request in the form of a workflow $f(G, d)$, which specifies a workflow structure G and a deadline d. The workflow structure is defined as a DAG $G(V, A)$, where each vertex $v_j \in V$ represents a component job, and each directed edge $a_{j,j'} \in A$ denotes an execution dependency, i.e. the actual finish time (AFT) t_j^{AF} of job v_j must not be later than the actual start time (AST) $t_{j'}^{AS}$ of job $v_{j'}$. 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.

MapReduce Model

We consider a MapReduce job v_j running a set of parallel map (or reduce) tasks, each of which requires a memory of size o_j and spends a percentage $\mu_{i,j}$ of time executing CPU-bound instructions on a CPU core of machine m_i and a percentage $(1-\mu_{i,j})$ of time executing I/O-bound instructions on machine m_i. In job v_j, generally, as the number K_j of parallel tasks increases, the workload $w_{j,k}(K_j)$ of each task $s_{j,k}$ decreases and the total workload $w_j(K_j) = K_j \cdot w_{j,k}(K_j)$ of all tasks increases. However, the maximum number K'_j of tasks that can be executed in parallel without performance degradation is limited by the cluster capacity, e.g. $K'_j \leq \sum_{m_i \in M} \min\{N_i, \lfloor o_i/o_j \rfloor\}$. Note that a serial program can be considered as a special case of a MapReduce job with $K'_j = 1$. The execution time of task $s_{j,k}$ on machine m_i is $t_{i,j,k} = w_{j,k}(K_j)/(\mu_{i,j} \cdot p_i)$. Estimating the execution time of a task on any service is an important issue. Many techniques have been proposed such as code analysis, analytical benchmarking/code profiling, and statistical prediction [31, 32], which are beyond the scope of this paper.

The active state $a_{i,j,k}(t)$ of task $s_{j,k}$ on machine m_i is 1 (or 0) if it is active (or inactive) at time t. The number of active tasks in job v_j on machine m_i at time t is $n_{i,j}(t) = \sum_{s_{j,k} \in v_j} a_{i,j,k}(t)$. The number of CPU cores and the size of memory used by all component jobs of a workflow on machine m_i at time t are $n_i(t) = \sum_{v_j \in V} n_{i,j}(t)$ and $o_i(t) = \sum_{v_j \in V} [o_j n_{i,j}(t)]$, respectively.

Energy Model

The DEC of a workflow in a cluster is $E = \sum_{m_i \in M} \{P_i \sum_{v_j \in V} [\mu_{i,j} \int_0^{t^C} n_{i,j}(t)dt]\}$, where P_i is the dynamic power consumption (DPC) of a fully utilized CPU core, and which is validated by energy measurements of practical systems in [14].

Mapping Function

We define a workflow mapping function as $\mathfrak{M} : \{s_k(v_j) \xrightarrow{[t^S_{j,k},\ t^E_{j,k}]} m_i, \forall v_j \in V, \exists m_i \in M, \exists [t^S_{j,k}, t^F_{j,k}] \subset T\}$, which denotes that the k-th task of the j-th job is mapped onto the i-th machine from time $t^S_{j,k}$ to time $t^E_{j,k}$. The domain of this mapping function spans across all possible combinations of a set V of component jobs of the workflow, a set M of machines, and a time period T of workflow execution.

Problem Definition

We formulate a deadline- and resource-constrained workflow mapping problem for energy efficiency (EEWM):

Table 1: Notations used in the cost models.

Notations	Definitions
$M = \bigcup_l C_l$	a cluster of machines divided into homogeneous subclusters $\{C_l\}$
$m_i(N_i, p_i, o_i, P_i)$	the i-th machine equipped with a memory of size o_i and N_i CPU cores of speed p_i and DPC P_i per core at full utilization
R	the available resource-time table of cluster M
$N_i^A(t)$	the number of idle CPU cores on machine m_i at time t
$o_i^A(t)$	the size of available memory on machine m_i at time t
$f(G(V, A), d)$	a workflow request consisting of a workflow structure of a DAG $G(V, A)$ and a deadline d
v_j, $s_{j,k}$	the j-th component job in a workflow and the k-th task in job v_j
$a_{j,j'}$	the directed edge from job v_j to job $v_{j'}$
t_j^{AS}, t_j^{AF}	the actual start and finish time of job v_j
t^C	the completion time of a workflow
$\mu_{i,j}$	the percentage of execution time for CPU-bound instructions in job v_j on machine m_i
o_j	the memory demand per task in job v_j
$w_j(K)$	the workload of job v_j partitioned into K tasks
$w_{j,k}(K)$	the workload of task $s_{j,k}$ in v_j with K tasks
K_j, K'_j	the number and the maximum possible number of tasks in v_j
$t_{i,j,k}$	the execution time of task $s_{j,k}$ running on machine m_i
$a_{i,j,k}(t)$	indicate whether task $s_{j,k}$ is active on machine m_i at time t
$n_{i,j}(t)$	the number of running tasks in job v_j on machine m_i at time t
$n_i(t)$	the number of CPU cores used by f on machine m_i at time t
$o_i(t)$	the size of memory used by workflow f on machine m_i at time t
E	the DEC of workflow f in cluster M

Definition 1. ***EEWM****: Given a cluster $\{m_i(N_i, p_i, o_i, P_i)\}$ of machines with an available resource-time table $\{N_i^A(t), o_i^A(t)\}$, and a workflow request $f(G(V, A), d)$, where each job v_j has a set $\{w_j(K_j)|K_j = 1, 2, \ldots, K'_j\}$ of workloads for different task partitions, and each task in job v_j has a percentage $\mu_{i,j}$ of execution time for CPU-bound instructions on machine m_i and a memory demand o_j, we wish to find a mapping function $\mathfrak{M} : (V, M, T) \rightarrow \{s_k(v_j) \xrightarrow{[t_{j,k}^S,\ t_{j,k}^E]} m_i\}$ to minimize the dynamic energy consumption:*

$$\min_{\mathfrak{M}} E,$$

subject to the following time and resource constraints:

$$t^C \leq d,$$
$$t_j^{AF} \leq t_{j'}^{AS}, \forall a_{j,j'} \in A,$$
$$n_i(t) \leq N_i^A(t), \forall m_i \in M,$$
$$o_i(t) \leq o_i^A(t), \forall m_i \in M.$$

Complexity Analysis

We first consider a special case of EEWM with a sufficiently large upper bound on dynamic energy consumption as follows: given 5 machines with sufficient memory and a single CPU core of speed p and DPC P_i at full utilization, and J independent serial jobs $\{v_j\}$ with CPU-burst workload w_j, does there exist a feasible non-preemptive scheduling scheme such that the makespan is no more than d? This special case has been proved to be strongly NP-hard in [4], so is the general EEWM problem, which, with a polynomially bounded objective function, has no FPTAS unless $P = NP$ [33].

Special Case: Pipeline-structured Workflow

We start with a special case with a Pipelined-structured workflow running on HOmogeneous machines (PHO). We prove it to be NP-complete and design an FPTAS to solve EEWM-PHO.

Generally, we may achieve more energy savings on an under-utilized cluster than on a fully-utilized cluster. Hence, the problem for a single pipeline-structured workflow is still valuable in real-life systems. The EEWM-PHO problem is defined as follows.

Definition 2. ***EEWM-PHO****: Given I idle homogeneous machines $\{m_i(N, p, o, P)\}$ and a workflow $f(G(V, A), d)$ containing a chain of J jobs, where each job v_j has a workload list $\{w_j(K_j)|K_j = 1, 2, \ldots, K'_j\}$, and each task in job v_j has a percentage μ_j of execution time for CPU-bound instructions and a memory demand o_j, does there exist a feasible mapping scheme such that DEC is no more than E?*

Complexity Analysis

We prove that EEWM-PHO is NP-complete by reducing the two-choice knapsack problem (TCKP) to it.

Definition 3. ***Two-Choice Knapsack****: Given J classes of items to pack in a knapsack of capacity H, where each class C_j $(j = 1, 2, \ldots, J)$ has two items and each item $r_{j,l}$ $(l = 1, 2)$ has a value $b_{j,l}$ and a weight $h_{j,l}$, is there a choice of exactly one item from each class such that the total value is no less than B and the total weight does not exceed H?*

The knapsack problem is a special case of TCKP when we put each item in the knapsack problem and a dummy item with zero value and zero weight together into a class. Since the knapsack problem is NP-complete, so is TCKP.

Theorem 1. *EEWM-PHO is NP-complete.*

Proof. Obviously, EEWM-PHO $\in NP$. We prove that EEWM-PHO is NP-hard by reducing TCKP to EEWM-PHO. Let $(\{C_j(b_{j,1}, h_{j,1}, b_{j,2}, h_{j,2})|1 \leq j \leq J\}, B, H)$ be an instance of TCKP. Without loss of generality, we assume that $b_{j,1} > b_{j,2}$ and $h_{j,1} > h_{j,2} > 0$. If $h_{j,1} < h_{j,2}$, $r_{j,1}$ would always be selected. If $h_{j,2} = 0$, we can always add $\tau > 0$ to $h_{j,1}$, $h_{j,2}$ and H such that $h_{j,2} > 0$.

We construct an instance of EEWM-PHO as follows. Let $I = 2$, $d = H$, $v_j = C_j$, $K'_j = 2$, $o_j = o$, $w_j(1) = h_{j,1}\mu_j p$, $w_j(2) = 2h_{j,2}\mu_j p$, $u_j = (B_j - b_{j,1})/(h_{j,1}P)$ and $E = \sum_{1\leq j\leq J} B_j - B$, where $B_j = (2h_{j,2}b_{j,1} - h_{j,1}b_{j,2})/(2h_{j,2} - h_{j,1})$. This process can be done in polynomial time.

Then, if job v_j only has one task, its execution time is $t_j(1) = w_j(1)/(\mu_j p) = h_{j,1}$, and its DEC is $E_j(1) = t_j(1)\mu_j P = B_j - b_{j,1}$. If job v_j has two tasks, the execution time of each task is $t_j(2) = w_j(2)/(2\mu_j p) = h_{j,2}$, and the DEC of job v_j is $E_j(2) = 2t_j(2)\mu_j P = B_j - b_{j,2}$. Obviously, two tasks in a job are mapped onto two machines simultaneously.

As a result, $\sum_{1\leq j\leq J} t_j(K_j) = \sum_{1\leq j\leq J} h_{j,K_j}$, which means $\sum_{1\leq j\leq J} t_j(K_j) \leq d \Leftrightarrow \sum_{1\leq j\leq J} h_{j,K_j} \leq H$. Similarly, $\sum_{1\leq j\leq J} E_j(K_j) = \sum_{1\leq j\leq J} (B_j - b_{j,K_j}) = \sum_{1\leq j\leq J} B_j - \sum_{1\leq j\leq J} b_{j,K_j} = E + B - \sum_{1\leq j\leq J} b_{j,K_j}$, which means that $\sum_{1\leq j\leq J} E_j(K_j) \leq E \Leftrightarrow \sum_{1\leq j\leq J} b_{j,K_j} \geq B$. Therefore, if the answer to the given instance of TCKP is Yes (or No), the answer to the constructed instance of EEWM-IJOM is also Yes (or No). Proof ends. □

Approximation Algorithm

We prove that EEWM-PHO is weakly NP-complete and design an FPTAS as shown in Alg. 1 by reducing this problem to the weakly NP-complete restricted shortest path (RSP) problem, which can then be solved using an FPTAS proposed in [34].

Given an instance of EEWM-PHO, we construct an instance of RSP according to the pipeline-structured workflow as follows. As illustrated in Fig. 1, the network graph $\mathbb{G}$ consists of $\mathbb{V} = \{v_{j,k}|j = 1, \ldots, J, k = 1, \ldots, K'_j\} \cup \{u_0, u_j|j = 1, \ldots, J\}$ with a source u_0 and a destination u_J, and $\mathbb{E} = \{e_{2j-1,k}, e_{2j,k}|j = 1, \ldots, J, k = 1, \ldots, K'_j\}$, where $e_{2j-1,k} = (u_{j-1}, v_{j,k})$ and $e_{2j,k} = (v_{j,k}, u_j)$. Then, we calculate the execution time of job v_j with k tasks as $t_j(k) = w_j(k)/(k \cdot p \cdot \mu_j)$, and accordingly its DEC as $E_j(k) = k \cdot P \cdot \mu_j \cdot t_j(k)$. Subsequently, we assign the cost $c(e)$ and delay $l(e)$ of each edge $e \in \mathbb{E}$ as $c(e_{2j-1,k}) = E_j(k)$, $l(e_{2j-1,k}) = t_j(k)$, and $c(e_{2j,k}) = l(e_{2j,k}) = 0$, and set the delay constraint on a path from u_0 to u_J to be d. As a result, the minimum cost in RSP is exactly the minimum DEC in EEWM-PHO, and if $v_{j,k}$ is on the solution path to RSP, the j-th job has k tasks. Based on Theorem 1 and the above reduction, we have

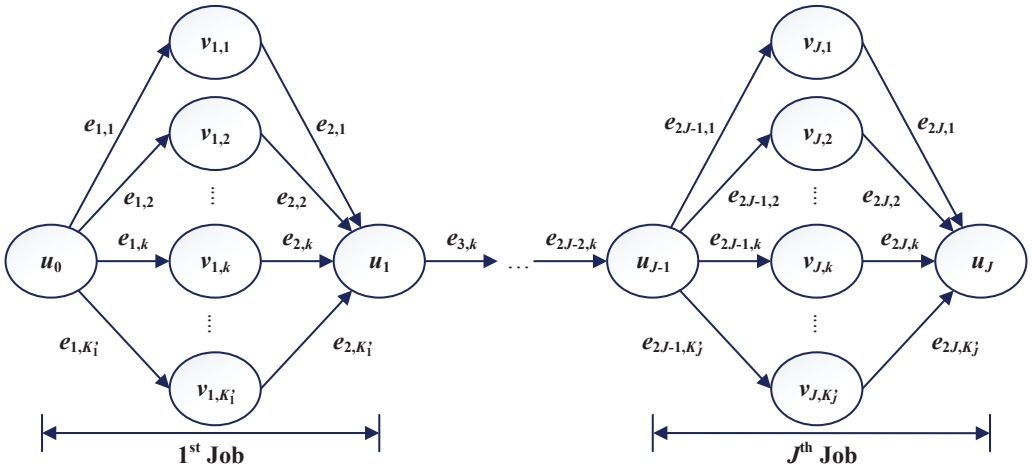


Figure 1: A constructed network corresponding to a workflow with a pipeline structure.

Algorithm 1: EEWM-PHO-FPTAS

Input: A cluster $\{m_i(N, p, o, P)\}$ and a chain of jobs $\{v_j\}$ with a deadline d and a set $\{w_j(K_j)\}$ of workloads

1: Construct a DAG $\mathbb{G}(\mathbb{V}, \mathbb{E})$ for pipeline $\{v_j\}$ as shown in Fig. 1, and assign energy cost $E_j(k)$ and delay $t_j(k)$ to edge $e_{2j-1,k}$ and zero cost and zero delay to edge $e_{2j,k}$;

2: Use FPTAS in [34] to find the minimum-cost path from u_0 to u_J under delay constraint d with approximate rate $(1+\epsilon)$ and convert it to mapping scheme.

Theorem 2. *EEWM-PHO is weakly NP-complete.*

Let $K' = \max_{1 \le j \le J} K_j'$. Then, $|\mathbb{V}| \le JK' + J + 1$ and $|\mathbb{E}| \le 2JK'$ in the constructed graph $\mathbb{G}$. It is obvious that the construction process can be done within time $O(JK')$. Therefore, EEWM-PHO finds a feasible solution that consumes energy within the least DEC multiplied by $(1+\epsilon)$ in time $O(J^2K'^2/\epsilon)$ if the FPTAS in [34] is used to solve RSP in acyclic graphs. Thanks to the special topology in Fig. 1, the time complexity is further reduced to $O(JK'(\log K' + 1/\epsilon))$.

Algorithm Design for an Arbitrary Workflow on a Heterogeneous Cluster

We consider EEWM with a DAG-structured workflow on a heterogeneous cluster and design a heuristic algorithm, referred to as big-data adaptive workflow mapping for energy efficiency (BAWMEE).

An Overview of BAWMEE

The key idea of BAWMEE is to partition a DAG into a set of pipelines and then repeatedly employ Alg. 1 with near optimality and low time complexity to achieve energy-efficient mapping of each pipeline.

In BAWMEE, each workflow mapping consists of two components: iterative critical path (CP) selection and pipeline mapping. A CP is the longest execution path in a workflow, which can be calculated in linear time. The algorithm starts with computing an initial CP according to the average execution time of each job running in serial on all the machines, 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 average execution time of a job as above and performs a pipeline mapping of the computed CP until there are no branches left.

In pipeline mapping, we consider two extreme scenarios: resource/time sufficiency and resource/time insufficiency. In the former case, we only need to focus on energy efficiency, while in the latter case, it may be unlikely to meet the performance requirement. Therefore, we design one algorithm for each of these two scenarios: a heuristic for energy-efficient pipeline mapping (EEPM) under a deadline constraint in Alg. 3, which calls Alg. 1, and a heuristic for minimum delay pipeline mapping (MDPM) with energy awareness in Alg. 4. If Alg. 3 fails to find a feasible mapping scheme due to limited resources, we resort to Alg. 4. In EEPM, due to the homogeneity of tasks in a job, we map all the tasks in the same job onto a homogeneous sub-cluster, hence using Alg. 1 to balance the trade-off between execution time and DEC (directly associated with total workload) for each job on a pipeline. In MDPM, we search for a good task partitioning to minimize the end time of each job through a limited number of tries by reducing the possible number of tasks in each job v_j from $\{1, 2, 3, \ldots, K'_j\}$ to $\{1, 2, 2^2, \ldots, 2^{\lfloor \log K'_j \rfloor}\} \cup \{K'_j\}$.

Algorithm Description

If a job v_j has been mapped, it has AST t_j^{AS} and AFT t_j^{AF}. If all the preceding (and succeeding) jobs, in $Prec$ (and $Succ$), of job v_j are mapped, its earliest start time (EST) (and LFT) can be calculated as

$$t_j^{ES} = \begin{cases} 0, \text{ if } v_j \text{ is the start job of workflow } f, \\ \max\limits_{v_{j'} \in Prec(v_j)} t_{j'}^{AF}, \text{ otherwise;} \end{cases}$$

Table 2: Time-Energy Table Tbl_j of Job v_j.

$t_j(K_{j,1}, C_{j,1})$	$<$	$t_j(K_{j,2}, C_{j,2})$	$<$	$\ldots$	$<$	$t_j(K_{j,n}, C_{j,n})$
$E_j(K_{j,1}, C_{j,1})$	$>$	$E_j(K_{j,2}, C_{j,2})$	$>$	$\ldots$	$>$	$E_j(K_{j,n}C_{j,n})$
$K_{j,1} \in [1, K'_j]$		$K_{j,2} \in [1, K'_j]$		$\ldots$		$K_{j,n} \in [1, K'_j]$
$C_{j,1} \subset M$		$C_{j,2} \subset M$		$\ldots$		$C_{j,n} \subset M$

and

$$t_j^{LF} = \begin{cases} d, \text{ if } v_j \text{ is the end job of workflow } f, \\ \min\limits_{v_{j'} \in Succ(v_j)} t_{j'}^{AS}, \text{ otherwise,} \end{cases}$$

respectively. If there exist unmapped preceding and succeeding jobs of v_j, its temporary earliest start time (TEST) $t'_{ES}(v_j)$ and temporary last finish time (TLFT) $t'_{LF}(v_j)$ 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.

Each job v_j is associated with a set of pairs of the number $K_{j,n}$ of tasks and the used homogeneous sub-cluster $C_{j,n}$. Each pair corresponds to a certain execution time $t_j(K_{j,n}, C_{j,n})$ and DEC $E_j(K_{j,n}, C_{j,n}) = P(C_{j,n})w_j(K_{j,n})/p(C_{j,n})$, where $p(C_{j,n})$ and $P(C_{j,n})$ are the speed and the DPC of a fully utilized CPU core on a machine in $C_{j,n}$, respectively, and $w_j(K_{j,n})$ is the workload of v_j with $K_{j,n}$ tasks. All the quadruples $\{(t_j(K_{j,n}, C_{j,n}), E_j(K_{j,n}, C_{j,n}), K_{j,n}, C_{j,n})\}$ are sorted in the ascending order of execution time as listed in Table 2, and are referred to as the time-energy table (TET) Tbl_j of job v_j. Any quadruple with both execution time and DEC larger (worse) than those of another will be deleted from Tbl_j.

In Alg. 2, BAWMEE first builds a time-energy table for each job by calling $buildTET()$ (in Line 1). If the workflow cannot meet its deadline with each job running the fastest, BAWMEE performs energy-aware job mapping (EAJM) with minimum finish time for each job in a topologically sorted order by calling $simplyMap()$ (in Line 2). Otherwise, BAWMEE employs iterative CP selection to find a CP with the earliest LFT from unmapped jobs (in Line 8), and performs EEPM or MDPM (if EEPM fails) for the selected CP (in Lines 9-10), where EEPM and MDPM are described later in Algs. 3 and 4, respectively. If there is any job that cannot be mapped in MDPM, we cancel the mapping of its downstream jobs (in Lines 11-14). If it is the last job of the workflow, we perform EAJM with minimum finish time (in Lines 15-16).

In Alg. 3 of EEPM, we reset the EST for the input pipeline according to the earliest time such that enough resources are made available to the first job (in Lines 2-3). If the pipeline cannot meet its LFT with each job running the fastest, we exit EEPM (in Lines 4-5); otherwise, the mapping of

Algorithm 2: BAWMEE

Input: a workflow $f(G(V, A), d)$ and an ART table R for sub-clusters $\{C_l\}$

1: $Tbl \leftarrow buildTET(V, \{C_l\})$;
2: **if** $simplyMap(f, R(\{C_l\}), Tbl)$ =True **then**
3: **return** .
4: $t_j^{LF} \leftarrow +\infty$ for $\forall v_j \in f$; $t_J^{LF} \leftarrow d$ for the end job v_J in f;
5: Calculate the average execution time $\bar{t}_j$ of each job v_j running in serial on all the machines;
6: $G' \leftarrow G$;
7: **while** $\exists$ an unmapped job $\in V$ **do**
8: Find the critical path cp ending at a job v with the earliest LFT in G' according to $\{\bar{t}_j | v_j \in G'\}$;
9: **if** $EEPM(cp, R(\{C_l\}), Tbl)$ =False **then**
10: $v \leftarrow MDPM(cp, R(\{C_l\}))$;
11: **if** $v \neq Null$ **then**
12: $D \leftarrow$ {all the downstream jobs of v in $G - G'$};
13: **if** $D \neq \emptyset$ **then**
14: Cancel the mapping of each job $v' \in D$, and add v' and its associated precedence constraints to G';
15: **if** v is the last job of f **then**
16: $EAJM(v, R(\{C_l\}))$;
17: $G' \leftarrow G' - \{v_j \in cp | v_j$ is mapped$\}$;

a pipeline with its EST and LFT is converted into the RSP problem with a relaxed resource limit (in Line 6). Accordingly, we calculate the number of tasks, the sub-cluster, and the start/finish time for each job using Alg. 1 (in Line 7). Then, we check if the start and finish time of each job are between its TEST and TLFT in their execution order (in Lines 8-9). If there exists a job that violates the precedence constraint, we divide the pipeline at this job, and use Alg. 3 to compute the mapping of the upstream sub-pipeline with an updated LFT constraint (in Lines 10-15). We repeat this process until we find a sub-pipeline whose mapping meets all precedence constraints. If the cluster is able to provide each job in this sub-pipeline with enough computing resources based on the mapping result of Alg. 1, we proceed with this mapping (in Lines 16-18); otherwise, we fail to find an EEPM and thus exit (in Line 19). In this case, BAWMEE would proceed to search for an MDPM.

Algorithm 3: EEPM

Input: a pipeline pl with its EST $pl.est$ and LFT $pl.lft$, an ART table $R(\{C_l\})$, and TETs $\{Tbl_j\}$

Output: a boolean variable to indicate whether pl or its part is mapped

Label the index j of each job in pl from 1 to the length of pl;
Calculate the earliest possible start time of the first job in pl on any machine as est according to $R(\{C_l\})$;
$pl.est \leftarrow \max\{est, pl.est\}$;
if $\sum_{v_j \in pl} t_j(K_{j,1}, C_{j,1}) > pl.lft - pl.est$ **then**
    return False.
Convert pipeline pl, where each quadruple in Tbl_j of each job $v_j \in pl$ corresponds to one of its mapping options, into a network graph in RSP;
Use Alg. 1 to calculate the number K_j of tasks, sub-cluster $C(v_j)$, and start and finish time, t_j^S and t_j^F, for each job v_j;
for $v_{j+1} \in pl$ **do**
    if $t_j^F > t'_{LF}(v_j)$ or $t_j^F < t'_{ES}(v_{j+1})$ **then**
        $pl(1,j).est \leftarrow pl.est$;
        if $t_j^F > t'_{LF}(v_j)$ **then**
            $pl(1,j).lft \leftarrow t'_{LF}(v_j)$;
        else
            $pl(1,j).lft \leftarrow \min\{t'_{ES}(v_{j+1}), t'_{LF}(v_j), pl.lft\}$;
        return $EEPM(pl(1,j), R(\{C_l\}), Tbl)$;
if $\exists$ K_j pairs of a CPU core and memory of size o_j in $R(C(v_j))$ for $\forall v_j \in pl$ **then**
    Map all K_j tasks onto $C(v_j)$ from t_j^S to t_j^F for $\forall v_j \in pl$;
    return True;
return False;

In Alg. 4 of MDPM, we search for the earliest finish time (EFT) of each job using EAJM in their execution order, and thus obtain the EFT of the entire pipeline. In Alg. 5 of EAJM with the minimum finish time under resource constraints, we exponentially relax the limit on the maximum number of tasks in a job to make a tradeoff between the optimality and the time complexity of EAJM.

Since the calculation of the earliest possible start time of the first job in EEPM takes time of $O(M'H)$ and the pipeline mapping in EEPM takes time of $O(JK'L[\log(K'L)+1/\epsilon])$, the time com-

Algorithm 4: MDPM

Input: a pipeline pl and an ART table R for $\{C_l\}$

Output: the first job that cannot be mapped

1: **for all** $v_j \in pl$ **do**
2: **if** $EAJM(v_j, R(\{C_l\})) > t'_{LF}(v_j)$ **then**
3: Cancel the mapping of job v_j;
4: **return** v_j;
5: **return** $Null$.

Algorithm 5: EAJM

Input: a job v_j and an ART table R for sub-clusters $\{C_l\}$

Output: the EFT t_j^{EF} of job v_j

1: Update the TEST $t'_{ES}(v_j)$; $t_j^{EF} \leftarrow +\infty$;
2: **for** $K \leftarrow 1, 2, 4, \ldots 2^{\lfloor \log K'_j \rfloor}, K'_j$ **do**
3: Calculate the EFT $t_j^{EF}(K)$ of job v_j with K tasks by minimizing the finish time of each task one by one;
4: **if** $t_j^{EF} > t_j^{EF}(K)$ **then**
5: $t_j^{EF} \leftarrow t_j^{EF}(K)$; $K_j \leftarrow K$;
6: Map job v_j consisting of K_j tasks until t_j^{EF};
7: **return** t_j^{EF}.

plexity of EEPM is $O(J^2K'L[\log(K'L)+1/\epsilon]+M'H)$. Since EAJM takes time of $O(M'HK'\log K')$, the time complexity of MDPM is $O(M'HJK'\log K')$. Therefore, the time complexity of BAWMEE is $O(JK'[JL(1/\epsilon+\log(K'L))+M'H\log K'])$. Here, M' is the number of machines; L is the number of homogeneous sub-clusters, J is the number of jobs; K' is the maximum number of tasks in a job; and H is the number of time slots in the ART table.

Numerical Examples

In this subsection, we use two simple examples to illustrate BAWMEE: one with sufficient resource and time, and the other with insufficient resource and time.

The first example considers an idle cluster $M = C_1 \cup C_2$ consisting of 4 single-core machines, where $C_1 = \{m_1, m_2\}$ and $C_2 = \{m_3, m_4\}$, and receives a workflow f comprised of homogeneous jobs organized in Fig. 2 with a deadline of 19 time units. The execution time and DEC of a

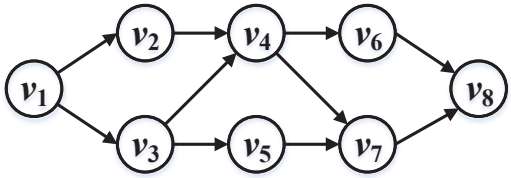


Figure 2: An example of a workflow structure G.

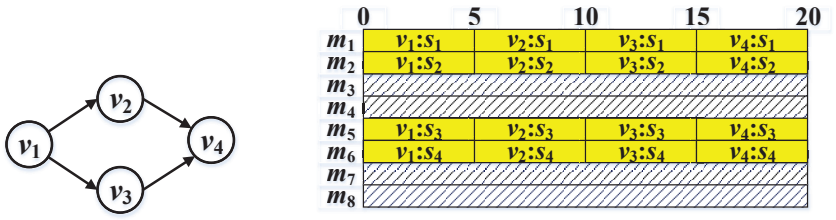
Table 3: Time-Energy Table in Example 1

Time	3	2	5	4		2	3	5
Energy	6	8	5	8	$\Rightarrow$	8	6	5
# of Tasks	1	2	1	2		2	1	1
Sub-cluster	C_1	C_1	C_2	C_2		C_1	C_1	C_2

(a) (b)

Figure 3: Workflow mapping in example 1: (a) BAWMEE; (b) Optimal.

job with a different task partitioning on a different sub-cluster are calculated and listed on the left side of Table 3. BAWMEE first builds a TET for each job on the right side of Table 3. A pipeline $\{v_1, v_2, v_4, v_6, v_8\}$ is selected as the initial CP. We assume that ϵ is set to be 0.02. In an approximation solution of pipeline mapping with EST of 0 and LFT of 19, each job has only one task, and v_1, v_2 and v_6 are mapped onto machine m_1 in C_1 from 0 to 3, from 3 to 6, and from 11 to 14, respectively, and v_4 and v_8 are mapped onto machine m_3 in C_2 from 6 to 11 and from 14 to 19, respectively. Then, the second pipeline $\{v_3, v_5, v_7\}$ is selected as the CP in $G - \{v_1, v_2, v_4, v_6, v_8\}$. In an approximation solution of pipeline mapping with EST of 3 and LFT of 14, v_3 intends to have one task and be mapped onto C_2 from 3 to 8, and v_5 and v_7 intend to have one task and be mapped onto C_1 from 8 to 11 and from 11 to 14, respectively. Since v_3 misses its TLFT of 6, the first sub-pipeline $\{v_3\}$ of $\{v_3, v_5, v_7\}$ is extracted and the approximation solution of sub-pipeline mapping with EST of 3 and LFT of 6 is that v_3 has one task and is mapped onto a machine m_2 in C_1 from 3 to 6. Subsequently, the third pipeline $\{v_5, v_7\}$ is selected as the CP in $G - \{v_1, v_2, v_3, v_4, v_6, v_8\}$, and the approximation solution of its mapping with EST of 6 and LFT of 14 is that v_5 intends to



(a) (b)

Figure 4: Example 2: (a) workflow structure; (b) workflow mapping.

Table 4: Time and Energy per Job in Example 2

Time	8	7	6	5	8	7	6	5
Energy	8	14	18	20	12	21	27	30
# of Tasks	1	2	3	4	1	2	3	4
Sub-cluster	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2

have one task and be mapped onto C_2 from 6 to 9 and v_7 intends to have one task and be mapped onto C_1 from 9 to 14. Since v_7 starts before its TEST of 11, the first sub-pipeline $\{v_5\}$ of $\{v_5, v_7\}$ is extracted and the approximation mapping solution of the sub-pipeline with EST of 6 and LFT of 11 is that v_5 has one task and is mapped onto a machine m_4 in C_2 from 6 to 11. Finally, the fourth pipeline $\{v_7\}$ is selected as the CP in $G - \{v_1, v_2, v_3, v_4, v_5, v_6, v_8\}$, and the approximation solution of its mapping with EST of 11 and LFT of 14 is that v_7 has one task and is mapped onto machine m_2 in C_2 from 11 to 14. Specifically, the mapping result of BAWMEE is shown in Fig. 3(a), and its DEC is 45 units. The optimal mapping is shown in Fig. 3(b), and the minimum DEC is 44 units.

The second example considers a cluster $M = C_1 \cup C_2$ consisting of 8 single-core machines, where $C_1 = \{m_1, m_2, m_3, m_4\}$ and $C_2 = \{m_5, m_6, m_7, m_8\}$, and m_3, m_4, m_7 and m_8 are busy and occupied by previous workflows. A user request specifies a workflow f comprised of homogeneous jobs organized in Fig. 4(a) with a deadline of 15 time units. The execution time and DEC of a job with a different task partitioning on a different sub-cluster are calculated and listed in Table 4. A pipeline $\{v_1, v_2, v_4\}$ is selected as the initial CP. EEPM intends to perform pipeline mapping with EST of 0 and LFT of 15 by partitioning each job of v_1, v_2 and v_4 into 4 tasks and mapping them onto C_1. However, C_1 does not have enough resources to support this mapping. Due to the failure of EEPM, MDPM attempts to partition each job into 1, 2, and 4 tasks to search for the minimum completion time of each job one by one. As a result, v_1, v_2 and v_4 are all partitioned

into 4 tasks, and mapped onto m_1, m_2, m_5 and m_6 from 0 to 5, from 5 to 10, and from 10 to 15, respectively. Then, the second pipeline $\{v_3\}$ with EST of 5 and LFT of 10 is selected as the CP in $G - \{v_1, v_2, v_4\}$, but fails to be mapped during time window $[5, 10]$ by EEPM and MDPM due to insufficient resources. Hence, BAWMEE cancels the mapping of the downstream mapped job $\{v_4\}$ of v_3, which is the first job that fails to be mapped before its TLFT of 10 by MDPM. Subsequently, the third pipeline $\{v_3, v_4\}$ with EST of 10 and LFT of 15 is selected as the CP in $G - \{v_1, v_2\}$, and fails to be mapped by EEPM. Thus, MDPM partitions v_3 into 4 tasks and maps them onto M from 10 to 15, but does nothing for v_4 due to missing its TLFT of 15. Finally, BAWMEE partitions the end job v_4 of the workflow into 4 tasks and maps them onto M from 15 to 20. Specifically, the mapping result of BAWMEE is shown in Fig. 4(b), and its DEC is 176 units.

Performance Evaluation

We conduct experiments to illustrate the effect of task partitioning on job workload and energy consumption, and evaluate the performance of EEWM-PHO-FPTAS in a practical setting for the special case of a pipeline-structured workflow on a homogeneous cluster in comparison with the default and optimal workflow mapping schemes. For the generalized problem, we conduct simulations to evaluate the performance of BAWMEE in comparison with three existing algorithms adapted from different scenarios: i) EEDAW in Alg. 7 adapted from a MapReduce job scheduling algorithm EEDAJ in Alg. 6 (integrated with the algorithms in [14] and [16]) by extending the progress estimation of a MapReduce job to that of a workflow, ii) MinD+ED adapted from a workflow scheduling algorithm with serial jobs in [6] by fixing the number of tasks in each MapReduce job and replacing preemptive task scheduling with non-preemptive task scheduling, and iii) MinD+EEDAJ comprised of the MinD algorithm in [6] for determining the virtual deadline of each job in a workflow and EEDAJ for scheduling MapReduce jobs onto energy-efficient machines before their virtual deadlines. In these three existing algorithms, we preset the number of tasks in each MapReduce job to be the maximum number of tasks to illustrate the benefits brought forth by the adaptive task partitioning strategy in our algorithm.

Experimental Settings

We set up a small-scale homogeneous cluster comprised of two Dell servers, each of which is equipped with 2 processors of Intel(R) Xeon(R) CPU E5-2630 v3 with 15MB cache and 6 cores of 2.4GHz, 16GB 2133MHz DDR4 RDIMM ECC memory, and 256GB 2.5inch serial ATA solid

Algorithm 6: EEDAJ()

Input: Unmapped jobs $\{v, d(v)\}$ and an available resource-time table R for a cluster M

while $Q_J \neq \emptyset$ **do**
  $v \leftarrow Q_J.top()$; // Q_J is a priority queue of all ready jobs. The priority of each job v is based on its deadline and execution progress, i.e. $rank(v) = d(v) - \sum_{s \in U(v)} \bar{t}(s)$, where $U(v)$ is a set of all unmapped tasks in v, and $\bar{t}(s)$ is the average execution time of task s on all machines.
  if v is ready **then**
    Select a task s from job v and estimate its expected finish time $d(s) = d(v) - [d(v) - t^{ES}(v)] \cdot (|U(v)| - 1)/|v|$, where $|v|$ is the number of tasks in v;
    Map task s to minimize incremental energy consumption before $d(s)$ or to minimize finish time if the former fails;
    if s is the last task in v **then**
      Update the AFT of v and the EST of all its succeeding jobs;
      $Q_J.dequeue()$;
  else
    Sleep for a period Δt; // $\Delta t = 6$ seconds

Figure 5: The experimental testbed for measuring energy consumption.

state drive. We install a power meter of 0.5% relative measurement errors with a measurement resolution of 1 watt, HOBO Plug Load Data Logger – UX120-018, to collect the active power/energy consumption of the entire cluster in the testbed, as shown in Fig. 5. The initial measurement shows that the total static power consumption of this mini-cluster in an idle state is 153.5 W on average.

On the cluster, we install Apache Hadoop 2.7.3 [35] and Oozie 4.3 [36], a workflow engine that automatically dispatches each component MapReduce job in a workflow with its respective configuration once all its preceding jobs finish. According to our Hadoop configuration, at most 23 map tasks can run in parallel. We download the airline on-time performance dataset of 11.2 GB for a period of 22 years (1987-2008) from the statistical computing website [37], and implement

Algorithm 7: EEDAW()

Input: Unmapped workflows $\{f(G(V, A), d)\}$ and an available resource-time table R for a cluster M

while $Q_W \neq \emptyset$ **do**
  $f \leftarrow Q_W.top()$; // Q_W is a workflow priority queue. The priority of each workflow f is based on its deadline $d(f)$ and execution progress, i.e. $rank(f) = d(f) - \sum_{s \in U(f)} \bar{t}(s)$, where $U(f)$ is a set of all unmapped tasks in f, and $\bar{t}(s)$ is the average execution time of task s on all machines.
  $v \leftarrow Q_J(f).first()$; // The jobs in the job queue $Q_J(f)$ of workflow f follow a topological sorting.
  Estimate the virtual deadline $d(v)$ of job v by $d(v) = t^{ES}(v) + [d(f) - t^{ES}(v)] \cdot \bar{t}(v)/[\bar{t}(v) + \sum_{v_j \in D(v)} \bar{t}(v_j)]$, where $\bar{t}(v) = \sum_{s \in v} \bar{t}(s)$.
  if v is ready **then**
    Select a task s from job v;
    Map task s to minimize incremental energy consumption before $d(v)$ or to minimize finish time if the former fails;
    if s is the last task in v **then**
      Update the AFT of v and the EST of all its succeeding jobs;
      $Q_J(f).dequeue()$;
      if $Q_J(f) = \emptyset$ **then**
        $Q_W.pop()$;
  else
    Sleep for a period Δt; // $\Delta t = 10$ minutes

three MapReduce programs to compute 1) the probability of each airline for being on schedule (PAS), 2) the average taxi in/out time per flight at each airport (ATA), and 3) the frequency of each flight cancellation reason (FCR). Initially, the dataset is stored in 22 separate files. To avoid block fragmentation in HDFS, we merge all the input data into a single file, and then upload the combined file into HDFS. In fact, the number of reduce keys would affect the parallelization degree of reduce tasks. The first MapReduce job (i.e. PAS) has 29 reduce keys (i.e. airport names); the second MapReduce job (i.e. ATA) has 340 reduce keys (i.e. flight numbers); and the third MapReduce job (i.e. FCR) has 5 reduce keys (i.e. cancellation codes).

Experimental Results

Performance Model

We consider a computing performance model where the total DEC, proportional to the total workload, of a moldable parallel job increases and the execution time of each task decreases as the number of tasks in the job increases. To validate this model, we first conduct experiments to

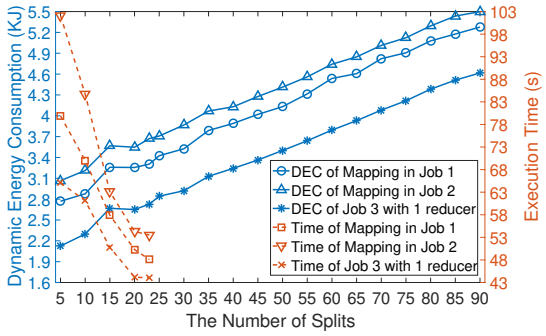


Figure 6: The DEC vs. the number of splits.

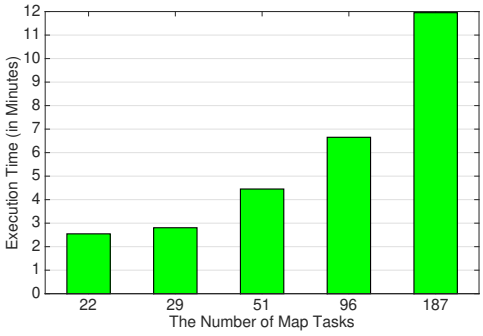


Figure 7: The execution time of a MapReduce job vs. the number of tasks.

illustrate the effect of task partitioning on job workload and DEC in big data applications, which lays down the foundation of this research.

Towards this goal, we repeatedly run each MapReduce program with and without the reducing phase for 10 times on our homogenous cluster to measure the DEC and execution time of the mapping and reducing phases of each MapReduce job, respectively. To adjust the number of mappers and reducers in each MapReduce job, we set the properties of "mapreduce.input.fileinputformat.split.minsize", "mapreduce.input.fileinputformat.split.maxsize", and "mapreduce.job.reduces" to be different values in the configuration file. To make the map tasks homogeneous, we divide the entire input data evenly by properly adjusting the split size. We tabulate the average DEC and execution time of the mapping and reducing phases of each MapReduce job with different numbers of tasks in Tables 5 and 6, respectively, where the DEC and execution time of mapping with different numbers of splits are further plotted in Fig. 6 for a visual comparison. These results show that the DEC of a MapReduce job increases while its execution time decreases as the number of map tasks increases up to 23, which is the largest number of map tasks supported simultaneously by the system. Such trend does not seem as obvious in reducing as in mapping for the following reasons. There exists a critical reduce-skew problem [38] for a small number of reduce keys. Also, since the workload of the reducing task is much less than that of the mapping task in our MapReduce jobs, the measurement errors for the reduce tasks in Table 6 are relatively larger in our measurement approach, where the DEC (or the execution time) of the reduce tasks is calculated as the difference between the DEC of the whole job with the default split size and that of the corresponding map-only job with the same split size.

Table 5: The Execution Time and DEC of Mapping vs. the Number of Splits.

The Number of Splits (or Mappers)	Split Size (MB)	Mapping in Job 1		Mapping in Job 2		Job 3 with 1 reducer[a]	
		Time (s)	DEC (KJ)	Time (s)	DEC (KJ)	Time (s)	DEC (KJ)
5	2250	79.9	2.770	102.1	3.066	65.1	2.125
10	1136	70.0	2.877	84.7	3.212	61.1	2.298
15	760	57.9	3.259	63.1	3.568	50.8	2.668
20	571	50.3	3.256	54.3	3.547	44.2	2.651
23	497	48.1	3.304	53.5	3.671	44.0	2.728
25	458	55.1	3.425	60.0	3.708	48.6	2.843
30	382	56.5	3.525	62.2	3.867	50.7	2.921
35	327	58.9	3.788	63.7	4.068	52.3	3.129
40	287	56.2	3.890	59.1	4.129	49.9	3.243
45	255	56.3	4.020	61.2	4.277	51.3	3.363
50	229	60.1	4.131	61.9	4.413	54.8	3.500
55	209	64.7	4.315	67.6	4.560	58.4	3.640
60	191	65.1	4.537	67.8	4.742	58.5	3.794
65	177	63.0	4.606	65.9	4.848	58.0	3.930
70	164	66.1	4.821	68.3	5.013	60.8	4.081
75	153	68.9	4.907	71.0	5.123	63.2	4.216
80	144	72.6	5.078	74.5	5.293	64.8	4.383
85	135	71.6	5.177	73.6	5.435	65.5	4.513
90	128	71.9	5.338	75.1	5.498	67.4	4.617

[a] Since the reducing workload of the third MapReduce job is negligible in comparison with its mapping workload, we list the DEC and execution time of the whole job with only a single reducer here.

Table 6: The Execution Time and DEC of Reducing vs. the Number of Reducers.

The Number of Reducers	Reducing in Job 1		Reducing in Job 2	
	Time (s)	DEC (KJ)	Time (s)	DEC (KJ)
1	60.9	0.726	38.9	0.636
2	42.0	0.780	24.3	0.657
3	43.2	0.791	24.8	0.676
4	34.7	0.792	20.5	0.695
5	30.9	0.816	18.7	0.713
6	26.4	0.838	17.2	0.753
7	21.4	0.849	17.5	0.773
8			14.9	0.785

To further validate the computing performance model in different scenarios, we run the third MapReduce program (i.e. FCR) with 22 separate input files in HDFS on another older computer server equipped with 2 processors of Intel(R) Xeon(R) CPU E5-2630 with 6 cores of 2.3GHz and 64GB memory. The program execution time is measured and plotted in Fig. 7, which shows that the execution time of this MapReduce job increases as the number of tasks increases when the

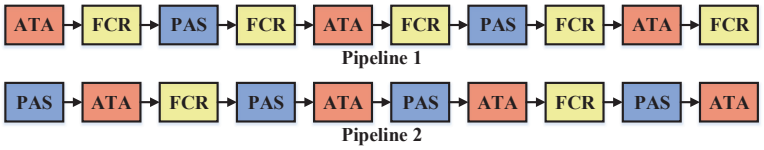


Figure 8: Pipeline-structured MapReduce workflows.

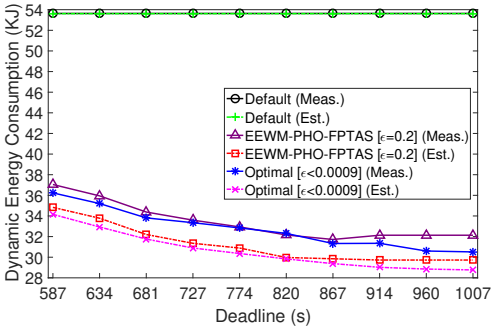


Figure 9: The DEC of Pipeline 1 under different deadline constraints.

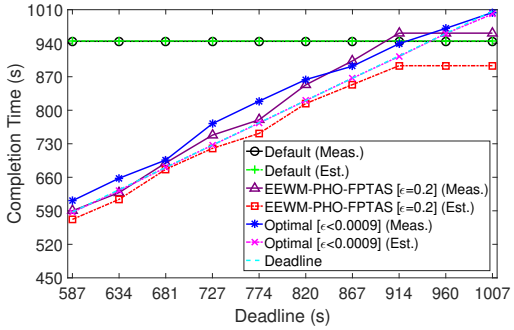


Figure 10: The completion time of Pipeline 1 under different deadline constraints.

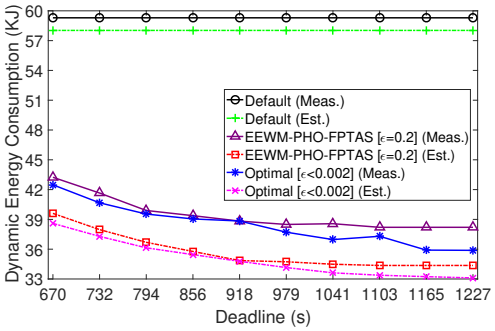


Figure 11: The DEC of Pipeline 2 under different deadline constraints.

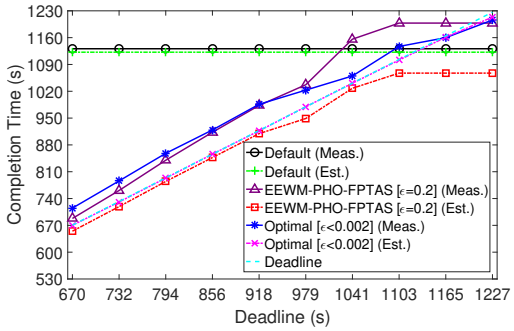


Figure 12: The completion time of Pipeline 2 under different deadline constraints.

server is fully utilized during the execution, which means that the total workload increases with the number of tasks.

Performance of EEWM-PHO-FPTAS

Although EEWM-PHO-FPTAS is designed for the special case of a pipeline-structured MapReduce workflow on a homogeneous cluster, it is the most important component of BAWMEE to solve the generalized problem. To test the practical performance of EEWM-PHO-FPTAS, we generate two pipeline-structured workflows comprised of 10 MapReduce jobs shown in Fig. 8, which

are randomly selected from the aforementioned three MapReduce programs. Since the existing energy-efficient MapReduce workflow mapping algorithms do not adjust the number of mappers and reducers, their workflow mapping schemes in this special case are exactly the same and completely rely on the default settings in Hadoop, where the number of mappers is the input size divided by the split size of 128 MB, and the number of reducers is 1. We conduct the workflow experiment on our homogeneous cluster, and plot in Figs. 9-12 the analytical estimations and experimental measurements of the DEC and completion time based on the workflow mapping scheme produced by EEWM-PHO-FPTAS, as well as the default and optimal workflow mapping schemes under 10 different deadline constraints. The experimental measurements show that EEWM-PHO-FPTAS with $\epsilon = 0.2$ cuts down 27% to 40% DEC at the cost of up to 6% more computing time in comparison with the default mapping scheme, and consumes only 6% more dynamic energy in comparison with the optimal mapping scheme. Hence, these results clearly illustrate the performance superiority of EEWM-PHO-FPTAS over existing energy-efficient workflow mapping algorithms in practice. Furthermore, we observe that the differences between the analytical estimations and the experimental measurements are less than 8% for the first pipeline and 11% for the second pipeline, which indicates the accuracy of our cost models in describing the main characteristics of workflow execution on a real Hadoop cluster. These discrepancies are mainly caused by ignoring the impact of the number of mappers and reducers on the execution time and DEC of shuffling in MapReduce jobs, and the measurement errors on reduce tasks.

Simulation Settings

To further evaluate the performance of the proposed heuristic for the generalized problem of larger scales, we conduct extensive simulations in various scenarios. We first generate a series of random workflows as follows: (i) randomly select the length L of the critical path of a workflow (no less than 3) and divide the workflow into L levels, in each of which every job has the same length of the longest path from the start job; (ii) randomly select the number of jobs in each level except the first and last levels, in which there is only one job; (iii) for each job, add an input edge from a randomly selected job in the immediately preceding level, if absent, and an output edge to a randomly selected job in its downstream level(s); (iv) randomly pick up two jobs in different levels and add a directed edge from the job in the upstream level to the job in the downstream level until we reach the given number of edges. The number of precedence constraints of the workflow is set to 1.5 times of the number of jobs, if possible. The maximum possible number of tasks for each

job is randomly selected between 12 and 48. The workload of a job is randomly selected between 0.6×10^{12} and 21.6×10^{12} CPU cycles when running in serial. According to the performance model of moldable jobs, the workload $w(k)$ of a job with $k > 1$ tasks is randomly selected between $w(k-1)[1 + 0.2/(k-1)]$ and $w(k-1)[1 + 0.6/(k-1)]$. We calculate the sum t_1 of the average execution time of the serial jobs on the critical path and the sum t_2 of the average execution time of all serial jobs according to the CPU speeds of all types of machines, and randomly select a workflow deadline baseline from the time range $[t_1, t_2]$. The percentage of execution time for the CPU-bound instructions of a task in each job on each type of machine is randomly selected from 0.6 to 1 at an interval of 0.1. By default, the amount of memory to request from the scheduler for each map/reduce task is 1GB in Hadoop/YARN. Based on our empirical study, we randomly select the memory demand of a task in each job from a range between 0.5GB and 4GB at an interval of 0.5GB.

We evaluate these algorithms in a heterogeneous cluster consisting of machines with four different specifications listed in Table 7, based on 4 types of Intel processors. Each homogeneous sub-cluster has the same number of machines. Each scheduling simulation lasts for 3 days and is repeated for 20 times with different workflow instances, whose arrivals follow the Poisson distribution. In the performance evaluation, each data point represents the average of 20 runs with a standard deviation. We set parameter ϵ in BAWMEE to be 0.2 to balance between workflow energy consumption and algorithm execution time. According to Figs. 9-12, when ϵ is set to be 0.2, the energy optimization performance is close to the optimal solution and BAWMEE is a polynomial-time solution. By default, the workflow size is randomly selected between 40 and 60 jobs; the cluster size and the average arrival interval of workflows are set to be 128 machines and 30 minutes, respectively; the deadline factor, which is a coefficient multiplied by the deadline baseline to determine the actual workflow deadline, is set to 0.15.

The dynamic energy consumption reduction (DECR) over the other algorithms in comparison is defined as

$$DECR(Other) = \frac{DEC_{Other} - DEC_{BAWMEE}}{DEC_{Other}} \cdot 100\%,$$

where DEC_{BAWMEE} and DEC_{Other} are the average DEC per workflow achieved by BAWMEE and the other algorithm, respectively. The deadline missing rate (DMR) is defined as the ratio of the number of workflows missing their deadlines to the total number of workflows. The unit running time (URT) is measured as the average simulation running time for computing the mapping scheme

Table 7: Specifications for Four Types of Machines.

Mach. Type	CPU Models	# of cores	Freq. (GHz)	DPC per core (W)	Mem. (GB)
1	6-core Xeon E7450	18	2.40	90	64
2	Single Core Xeon	6	3.20	92	64
3	2-core Xeon 7150N	12	3.50	150	64
4	Itanium 2 9152M	8	1.66	104	64

Table 8: Problem Sizes.

Index	$(\|V\|, \|M\|, 1/\lambda, T)$	Index	$(\|V\|, \|M\|, 1/\lambda, T)$
1	(3-7, 4, 240, 7)	11	(53-57, 192, 30, 1)
2	(8-12, 8, 200, 7)	12	(58-62, 256, 25, 1)
3	(13-17, 12, 160, 7)	13	(63-67, 384, 20, 1)
4	(18-22, 16, 150, 7)	14	(68-72, 512, 15, 1)
5	(23-27, 24, 120, 7)	15	(73-77, 768, 12, 1)
6	(28-32, 32, 105, 3)	16	(78-82, 1024, 10, 1/3)
7	(33-37, 48, 90, 3)	17	(83-87, 1536, 8, 1/3)
8	(38-42, 64, 60, 3)	18	(88-92, 2048, 6, 1/3)
9	(43-47, 96, 45, 3)	19	(93-97, 3072, 5, 1/3)
10	(48-52, 128, 30, 3)	20	(98-102, 4096, 4, 1/3)

of each workflow. The simulation runs on a Linux machine equipped with Intel Xeon CPU E5-2620 v3 of 2.4 GHz and a memory of 16 GB.

Simulation Results

Problem Size

For performance evaluation, we consider 20 different problem sizes from small to large scales, indexed from 1 to 20 as tabulated in Table 8. Each problem size is defined as a quadruple $(|V|, |M|, 1/\lambda, T)$, where $1/\lambda$ is the average arrival interval of workflow requests in minutes, and T is the time period in unit of days for accepting workflow requests in each simulation. As the workflow size and arrival frequency increase from index 1 to 20, we increase the resources correspondingly to meet tight deadlines with factor 0.15. We plot the DECR, DMR, and URT of EEDAW, MinD+ED, MinD+EEDAJ, and BAWMEE in Figs. 13-15, respectively, which show that BAWMEE saves 5.3% to 35.6%, 5.9% to 33.3%, and 6.3% to 34.5% DEC, and misses less deadlines in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively. Furthermore, the URT of BAWMEE is on the same order of magnitude as those of EEDAW, MinD+ED, and MinD+EEDAJ, and is less than 13 seconds even for problem index 20. We also plot the average number of tasks per job and average workload reduction of BAWMEE in Fig. 16, which sheds light on the energy efficiency of

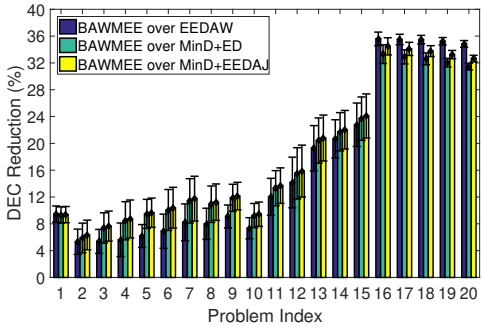


Figure 13: The DECR vs. problem sizes.

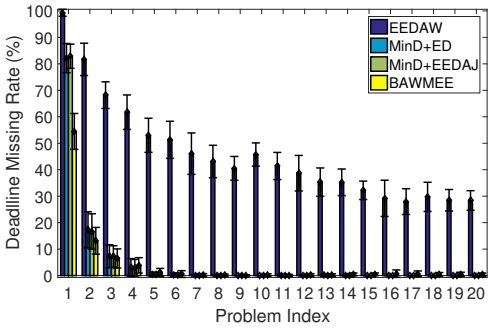


Figure 14: The DMR vs. problem sizes.

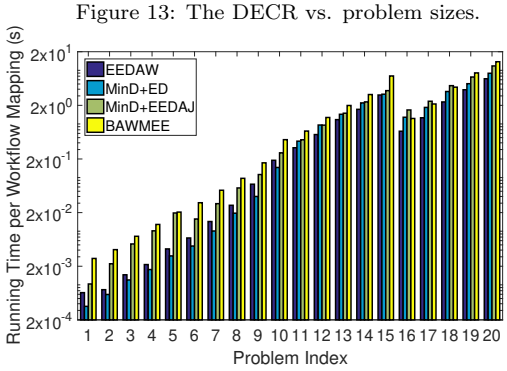


Figure 15: The URT vs. problem sizes.

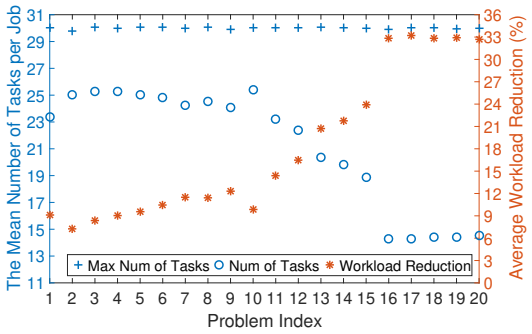


Figure 16: The adaptive task partitioning of BAWMEE vs. problem sizes.

BAWMEE. We observe from all the problem indices in Figs. 13 and 16 that on average a smaller number of tasks in each job would result in more reduced workload and thus more DEC reduction achieved by BAWMEE.

Deadline Constraint

We evaluate the performance of EEDAW, MinD+ED, MinD+EEDAJ, and BAWMEE in terms of DEC, DMR, and URT under different deadline constraints obtained from the deadline baseline multiplied by a factor from 0.05 to 1 with an interval of 0.05. The DEC, DMR, and URT of these algorithms are plotted in Figs. 17-19, respectively. These measurements show that BAWMEE saves up to 23.7%, 27.5%, and 28.2% DEC as the deadline increases in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively, and reduces DMR from 99.9% to 93.0% with a deadline factor of 0.05 and from 83.3% to 25.9% with a deadline factor of 0.1 compared to EEDAW. The DMR of BAWMEE is close to zero when the deadline factor is larger than 0.15, and is similar to those of MinD+ED and MinD+EEDAJ under various deadline constraints. Additionally, the

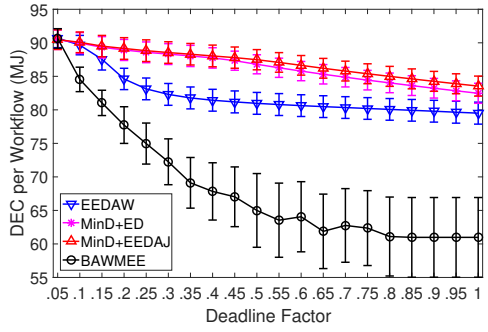


Figure 17: The DEC vs. deadlines.

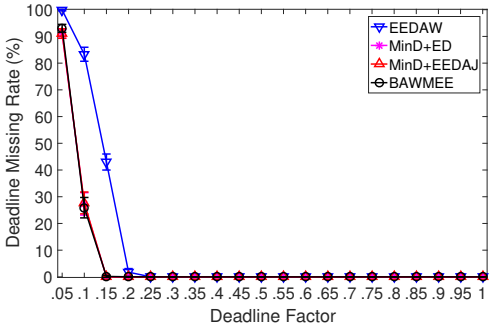


Figure 18: The DMR vs. deadlines.

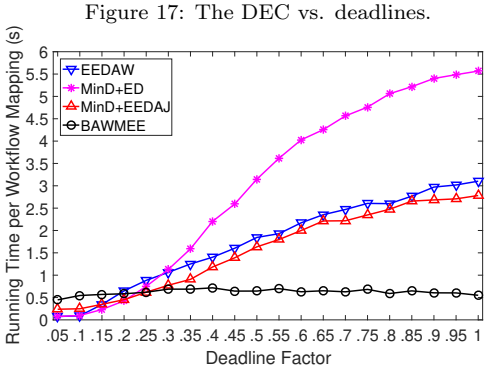


Figure 19: The URT vs. deadlines.

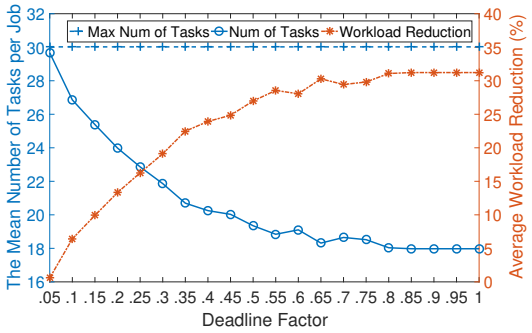


Figure 20: The adaptive task partitioning of BAWMEE vs. deadlines.

URT of BAWMEE is less than 0.7 second and is 17.7% to 5.9, 9.8% to 6.0, and 45.6% to 5.1 times of those of EEDAW, MinD+ED, and MinD+EEDAJ, respectively. It is worth pointing out that as the deadline increases, the DEC and URT of BAWMEE decrease, because EEPM plays a more significant role than MDPM in BAWMEE. We plot the average number of tasks per job and the average workload reduction of BAWMEE under different deadline constraints in Fig. 20, which clearly shows that BAWMEE reduces more workload overhead due to a decreased number of tasks as the deadline is relaxed, and explains why BAWMEE makes a better tradeoff between DEC and DMR than the other algorithms in comparison at an acceptable cost of running time.

Workflow Size

For scalability evaluation, we run these four algorithms under different average workflow sizes with 5 to 100 jobs per workflow at an interval of 5, where the maximum and minimum workflow sizes are 2 jobs more and less than the average workflow size, respectively. We plot the DECR, DMR, and URT of these algorithms in Figs. 21-23, respectively, where we observe that BAWMEE with

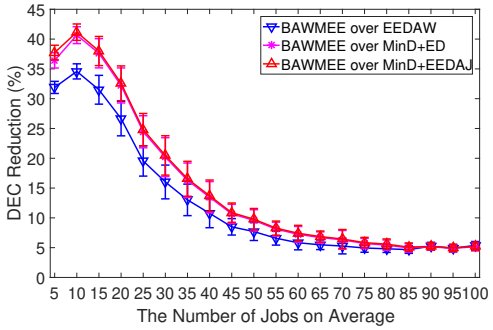


Figure 21: The DECR vs. workflow sizes.

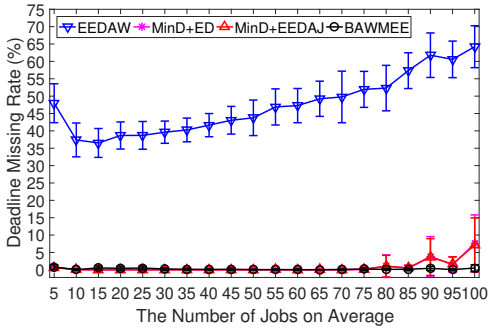


Figure 22: The DMR vs. workflow sizes.

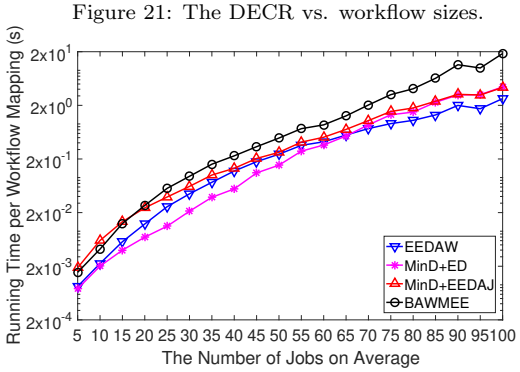


Figure 23: The URT vs. workflow sizes.

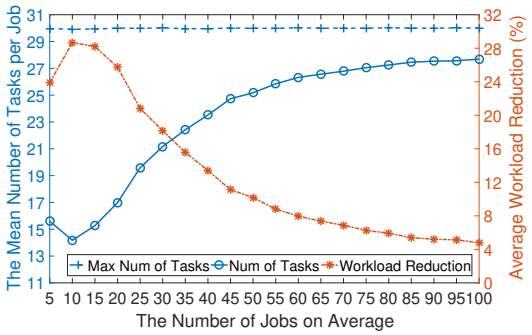


Figure 24: The adaptive task partitioning of BAWMEE vs. workflow sizes.

DMRs close to zero achieves an increasing DECR from 4.7% to 34.6%, from 4.9% to 40.7%, as well as from 5.0% to 41.2% in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively. For large workflow sizes with 50 to 100 jobs per workflow that impose high resource demands, BAWMEE achieves DECR only from 4.7% to 9.8%, because MDPM plays a more significant role than EEPM in BAWMEE, which is justified by the changes in the average number of tasks per job and the average workload reduction of BAWMEE plotted in Fig. 24. The DMR of EEDAW experiences a slump under the medium workflow sizes because a higher accuracy could be achieved in the execution progress of a smaller workflow than a larger one, while a further increase in the workflow size may lead to a more severe shortage of computing resources. In addition, the URT of BAWMEE is comparable with those of EEDAW, MinD+ED, and MinD+EEDAJ.

Cluster Size

We run these four algorithms under different cluster sizes of 64 to 256 machines at a step of 16 for scalability test. The DEC, DMR, and URT of these algorithms are plotted in Figs. 25-27, respec-

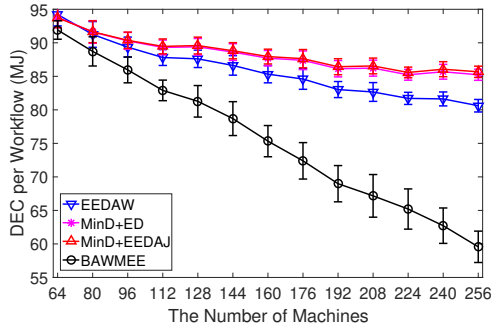


Figure 25: DEC vs. cluster sizes.

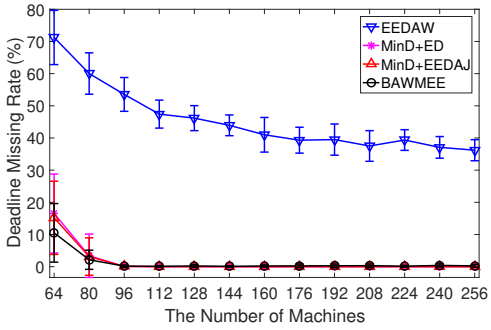


Figure 26: The DMR vs. cluster sizes.

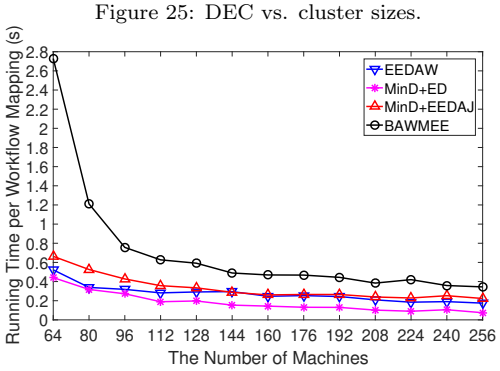


Figure 27: The URT vs. cluster sizes.

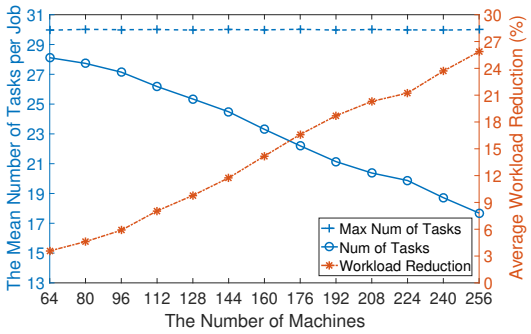


Figure 28: The adaptive task partitioning of BAWMEE vs. cluster sizes.

tively, where we observe that as the number of machines increases, BAWMEE consumes 2.5% to 26.1%, 2.0% to 30.1%, and 1.9% to 30.5% less DEC than EEDAW, MinD+ED, and MinD+EEDAJ, respectively, hence exhibiting a satisfactory scalability property with respect to the cluster size. Furthermore, the DMR of DAWMEE is only between 0.1% and 10.5% and is similar to those of MinD+ED and MinD+EEDAJ, while EEDAW misses 36.2% to 71.2% deadlines. The increase in the cluster size results in a relatively looser deadline and a more flexible workflow mapping, as a result of which, the DEC and DMR of these four algorithms decrease, and BAWMEE has more chances to save energy, which is consistent with the changes in the average number of tasks per job and the average workload reduction of BAWMEE plotted in Fig. 28. Moreover, the URT of BAWMEE is less than 2.7 seconds and is comparable with those of EEDAW, MinD+ED, and MinD+EEDAJ.

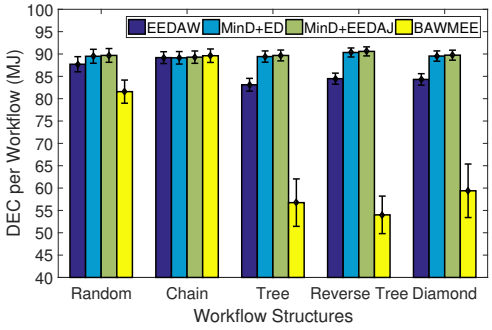


Figure 29: The DEC vs. workflow structures.

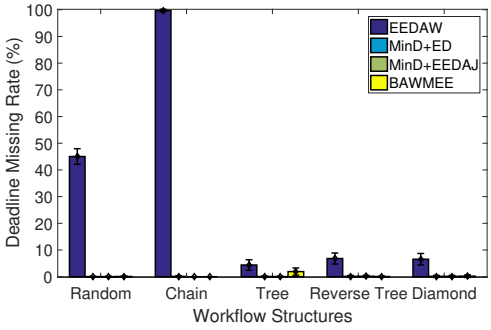


Figure 30: The DMR vs. workflow structures.

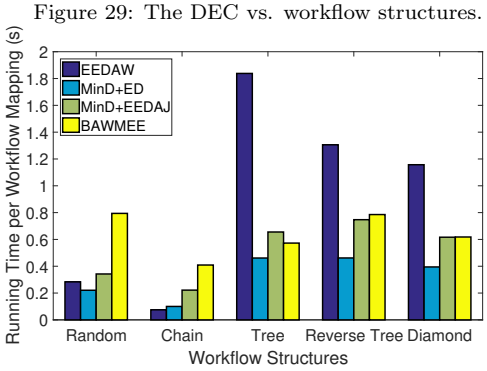


Figure 31: The URT vs. workflow structures.

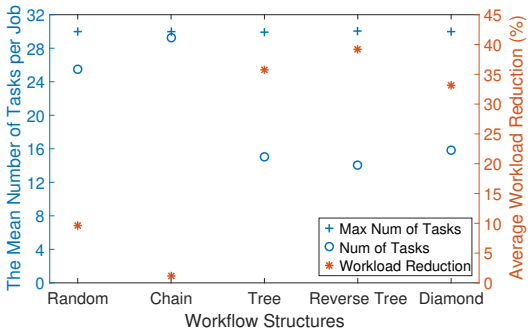


Figure 32: The adaptive task partitioning of BAWMEE vs. workflow structures.

Workflow Structure

We further investigate these four algorithms with various workflow structures, including a random shape, a chain, a tree, a reverse tree, and a diamond. The DEC, DMR, and URT are plotted in Figs. 29-31, respectively, which show that BAWMEE reduces DEC by 6.9% to 9.0%, 31.7% to 36.7%, 36.1% to 40.4%, and 29.6% to 33.8% in comparison with the other three algorithms in random, tree, reverse tree and diamond structured workflows, respectively. Here, BAWMEE fails to save energy in chain-structured workflows, because the deadline baseline is set too tight for this structure based on our deadline generation method, as indicated by the average number of tasks per job and the average workload reduction of BAWMEE in Fig. 32. BAWMEE almost misses no deadlines except in tree-structured workflows, where it favors the jobs close to the root more than those close to leaves and thus leads to an unfair division of the slack time [6]. Besides, the URT of BAWMEE is less than 0.8 seconds, and is 31.2% to 5.4, 1.2 to 4.1, and 87.4% to 2.3 times of those of EEDAW, MinD+ED, and MinD+EEDAJ with different workflow structures, respectively.

Conclusion

We investigated the property of moldable jobs and formulated a workflow mapping problem to minimize dynamic energy consumption under deadline and resource constraints. We designed an FPTAS for a special case with a pipeline-structured workflow on a homogeneous cluster, which we proved to be weakly NP-complete, and a heuristic for a generalized problem with an arbitrary workflow on a heterogeneous cluster. The performance superiority of the proposed solution in terms of dynamic energy saving and deadline missing rate was illustrated by extensive simulation results in comparison with existing algorithms, and further validated by real-life workflow implementation and experimental results using the Oozie workflow engine in the Hadoop/YARN ecosystem.

Acknowledgments

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-1526134 with New Jersey Institute of Technology.

References

References

[1] T. Shu, C. Wu, Energy-efficient mapping of big data workflows under deadline constraints, in: Proc. of Workshop on Workflows in Support of Large-Scale Science in conjunction with ACM/IEEE Supercomputing Conference, Salt Lake City, UT, USA, 2016, pp. 34–43.
URL http://ceur-ws.org/Vol-1800/paper5.pdf

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

[3] T. F. Gonzalez (Ed.), Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC, 2007.

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

[5] S. Abrishami, M. Naghibzadeh, D. Epema, Cost-driven scheduling of grid workflows using partial critical paths, IEEE TPDS 23 (8) (2012) 1400–1414.

[6] M. Zotkiewicz, M. Guzek, D. Kliazovich, P. Bouvry, Minimum dependencies energy-efficient scheduling in data centers, IEEE TPDS 27 (12) (2016) 3561–3574.

[7] J. L. amd C. Kozyrakis, On the energy (in)efficiency of Hadoop clusters, ACM SIGOPS Operating Systems Review 44 (1) (2010) 61–65.

[8] W. Lang, J. Patel, Energy management for MapReduce clusters, Proceedings of the VLDB Endowment 3 (1) (2010) 129–139.

[9] H. Amur, J. Cipar, V. Gupta, G. Ganger, M. Kozuch, K. Schwan, Robust and flexible power-proportional storage, in: Proc. of ACM SoCC, Indianapolis, IN, USA, 2010, pp. 217–228.

[10] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz, Energy efficiency for large-scale MapReduce workloads with significant interactive analysis, in: Proc. of ACM EuroSys, Bern, Switzerland, 2012, pp. 43–56.

[11] E. Bampis, V. Chau, D. Letsios, G. Lucarelli, I. Milis, G. Zois, energy Efficient Scheduling of MapReduce Jobs. https://arxiv.org/pdf/1402.2810.pdf (2014).

[12] M. Cardosa, A. Singh, H. Pucha, A. Chandra, Exploiting spatio-temporal tradeoffs for energy-aware MapReduce in the cloud, IEEE Tran. on Computers 61 (12) (2012) 1737–1751.

[13] L. Mashayekhy, M. Nejad, D. Grosu, Q. Zhang, W. Shi, Energy-aware scheduling of MapReduce jobs for big data applications, IEEE TPDS 26 (10) (2015) 2720–2733.

[14] D. Cheng, P. Lama, C. Jiang, X. Zhou, Towards energy efficiency in heterogeneous Hadoop clusters by adaptive task assignment, in: Proc. of IEEE ICDCS, Columbus, OH, USA, 2015, pp. 359–368.

[15] I. Goiri, K. Le, T. Nguyen, J. Guitart, J. Torres, R. Bianchini, GreenHadoop: Leveraging green energy in data-processing frameworks, in: Proc. of ACM EuroSys, Bern, Switzerland, 2012, pp. 57–70.

[16] D. Cheng, J. Rao, C. Jiang, X. Zhou, Resource and deadline-aware job scheduling in dynamic Hadoop clusters, in: Proc. of IEEE IPDPS, Hyderabad, India, 2015, pp. 956–965.

[17] B. Sharma, T. Wood, C. Das, HybridMR: A hierarchical mapreduce scheduler for hybrid data centers, in: Proc. of IEEE ICDCS, Philadelphia, PA, USA, 2013, pp. 102–111.

[18] J. Li, C. Pu, Y. Chen, V. Talwar, D. Milojicic, Improving preemptive scheduling with application-transparent checkpointing in shared clusters, in: Proc. of ACM Middleware, Vancouver, BC, Canada, 2015, pp. 222–234.

[19] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, R. Sakellariou, Energy-constrained provisioning for scientific workflow ensembles, in: Proc. of IEEE International Conference on Cloud and Green Computing, Karlsruhe, Germany, 2013, pp. 34–41.

[20] X. Xu, W. Dou, X. Zhang, J. Chen, Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment, IEEE Tran. on Cloud Comp. 4 (2) (2016) 166–179.

[21] Q. Zhu, J. Zhu, G. Agrawal, Power-aware consolidation of scientific workflows in virtualized environments, in: Proc. of ACM/IEEE SC, New Orleans, LA, USA, 2010, pp. 1–12.

[22] Y. Lee, H. Han, A. Zomaya, M. Yousif, Resource-efficient workflow scheduling in clouds, Elsevier Knowledge-Based Systems 80 (2015) 153–162.

[23] Y. Lee, A. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE TPDS 22 (8) (2011) 1374–1381.

[24] K. Li, Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers, IEEE Tran. on Computers 61 (12) (2012) 1668–1681.

[25] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Elsevier Info. Sci. 319 (2015) 113–131.

[26] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Elsevier Info. Sci. 379 (2017) 241–256.

[27] V. Nagarajan, J. Wolf, A. Balmin, K. Hildrum, FlowFlex: Malleable scheduling for flows of MapReduce jobs, in: Proc. of ACM/IFIP/USENIX Middleware, Beijing, China, 2013, pp. 103–122.

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

[29] K. Makarychev, D. Panigrahi, Precedence-constrained scheduling of malleable jobs with preemption, in: Proc. of ICALP, Copenhagen, Denmark, 2014, pp. 823–834.

[30] P. Sanders, J. Speck, Energy efficient frequency scaling and scheduling for malleable tasks, in: Proc. of Euro-Par, Rhodes Island, Greece, 2012, pp. 167–178.

[31] F. A. S. Verboven, P. Hellinckx, J. Broeckhove, Runtime prediction based grid scheduling of parameter sweep jobs, in: Proc. of IEEE Asia-Pacific Services Computing Conference, Taiwan, 2008, pp. 33–38.

[32] J. Shanthini, K. Shankarkumar, Anatomy study of execution time predictions in heterogeneous systems, International Journal of Computer Applications 45 (7) (2012) 39–43.

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

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

[35] Apache Hadoop. http://hadoop.apache.org (2016).

[36] Apache Oozie. https://oozie.apache.org (2016).

[37] Statistical Computing. http://stat-computing.org/dataexpo/2009/the-data.html (2009).

[38] Y. Kwon, M. Balazinska, B. Howe, J. Rolia, SkewTune: Mitigating skew in mapreduce applications, in: Proc. of ACM SIGMOD, Scottsdale, AZ, USA, 2012, pp. 25–36.