Energy-efficient Mapping of Big Data Workflows under Deadline Constraints *

Tong Shu and Chase Q. Wu
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
{ts372, chase.wu}@njit.edu

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. 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 Hadoop/YARN in comparison with existing algorithms.

Keywords

Big Data; Workflow Mapping; Green Computing

1 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, 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 programs. Energy efficiency of big data workflows in Hadoop systems 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 energy 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 [11]. 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 [15]. 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 [12]. 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 method [4].

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.

*Copyright held by the author(s).

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

- 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.
- We prove a deadline-constrained pipeline-structured workflow mapping problem for minimum total (energy) cost to be weakly NP-complete and design an FPTAS for it.
- 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 big data 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.

2 Related Work

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 [16, 5, 8]. Our research on job scheduling is orthogonal to these efforts, and adds an additional level of energy efficiency to Hadoop systems.

2.1 Energy-efficient Job Scheduling in Hadoop

2.1.1 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 [6]. 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) [22]. 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 [9].

2.1.2 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 [14]. 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 [10].

2.1.3 Resource Allocation

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 [19]. 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.

2.2 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 by migrating active VMs onto energy-efficient PMs in time [25] or consolidating applications with complementary resource requirements [28]. 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 [28].

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 [17], and designed two static energy-conscious workflow scheduling algorithms based on DVFS in heterogeneous distributed systems [18]. In [20], 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 [27], and designed a Pareto-based bi-objective genetic algorithm to achieve low energy consumption and high system reliability for static workflow scheduling [26].

2.3 Moldable/Malleable Job Scheduling

Some efforts have been made to minimize the completion time of a workflow comprised of malleable jobs [23, 7, 21], 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 [24].

3 Problem Formulation

3.1 Cost Models

3.1.1 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.

3.1.2 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.

3.1.3 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. 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, which are beyond the scope of this work.

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.

3.1.4 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 [9].

3.1.5 Mapping Function

We define a workflow mapping function as $\mathfrak{M} : \{s_k(v_j) \xrightarrow{[t_{j,k}^S,\ t_{j,k}^E]} m_i, \forall v_j \in V, \exists m_i \in M, \exists [t_{j,k}^S, t_{j,k}^F] \subset T\}$, which denotes that the k-th task of the j-th job is mapped onto the i-th machine from time $t_{j,k}^S$ to time $t_{j,k}^E$. The domain of this mapping function covers 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.

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

3.2 Problem Definition

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

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 deadline and resource constraints:

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

4 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?*

4.1 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 $(\{C_j(b_{j,1},h_{j,1},b_{j,2},h_{j,2})|1\le j\le J\},B,H)$ 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 $E=\sum_{1\le j\le J}B_j-B$, where $B_j=(2h_{j,2}b_{j,1}-h_{j,1}b_{j,2})/(2h_{j,2}-h_{j,1})$. It is obvious that the construction 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.

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. □

4.2 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 [13], which is solvable with an FPTAS.

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 $t_j(k)=w_j(k)/(k\cdot p\cdot\mu_j)$,

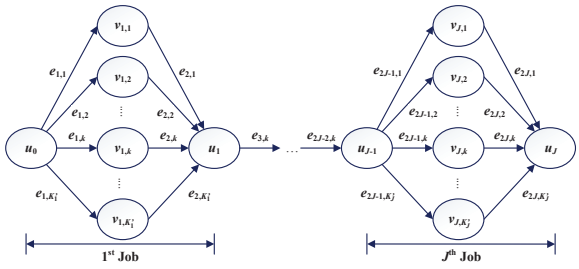

Figure 1: A constructed network corresponding to a workflow with a pipeline structure.

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 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 [13] 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.

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

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 [13] is used to solve RSP in acyclic graphs. Thanks to the special topology in Fig. 1, the time complexity is further reduced to $O(JK'(\log K'+1/\epsilon))$.

5 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).

5.1 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

Algorithm 2: BAWMEE

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

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

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

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

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;

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 $\{1, 2, 3, \dots, K'_j\}$ to $\{1, 2, 2^2, \dots, 2^{\lfloor \log K'_j \rfloor}\} \cup \{K'_j\}$.

5.2 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

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

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 $buildTET()$. 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 $simplyMap()$. Otherwise, BAWMEE employs iterative CP selection to find a CP with the earliest LFT from unmapped jobs, and performs EEPM or MDPM (if EEPM fails) for the selected CP. If there is any job that cannot

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}.

be mapped in MDPM, we cancel the mapping of its downstream jobs. If it is the last job of the workflow, we perform EAJM with minimum finish time.

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. If the pipeline cannot meet its LFT with each job running the fastest, we exit EEPM; otherwise, the mapping of a pipeline with its EST and LFT is converted into the RSP problem with a relaxed resource limit. Accordingly, we calculate the number of tasks, the sub-cluster, and the start/finish time for each job using Alg. 1. Then, we check if the start and finish time of each job are between its TEST and TLFT in their execution order. 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. 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 resources based on the mapping result of Alg. 1, we proceed with this mapping; otherwise, we fail to find an EEPM and thus exit. In this case, BAWMEE would proceed to search for an MDPM.

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 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 EEPM and MDPM are of $O(J^2K'L[\log(K'L) + 1/\epsilon] + M'H)$ and $O(M'HJK' \log K')$, respectively, the time complexity of BAWMEE is $O(J^2K'[JL(1/\epsilon + \log(K'L)) + M'H \log K'])$. 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.

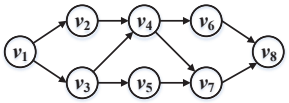

Figure 2: An example of a workflow structure G.

Table 3: Time-Energy Table in the Example

Time	3	2	5	4	⇒	2	3	5
Energy	6	8	5	8		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 the example: (a) BAWMEE; (b) Optimal.

5.3 A Numerical Example

In this subsection, we use a simple example to illustrate how BAWMEE achieves energy-efficient workflow mapping without violating precedence constraints. We consider 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 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 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 $G - \{v_1, v_2, v_3, v_4, v_5, v_6, v_8\}$,

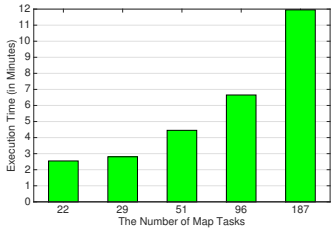

Figure 4: The execution time of a MapReduce job versus the number of tasks.

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.

6 Performance Evaluation

We conduct experiments to illustrate the effect of task partitioning on job workload and conduct simulations to evaluate the performance of BAWMEE in comparison with two existing algorithms adapted from different scenarios: i) pSciMapper adapted from a workflow mapping algorithm in [28] by applying the interference avoidance to MapReduce mapping, and ii) EEDA adapted from a MapReduce job mapping algorithm integrated with algorithms in [9] and [10].

6.1 Performance Model Validation

We consider a computing performance model where the total workload of a moldable parallel job increases and the execution time of each task decreases as the number of tasks increases. For model validation, we conduct an experiment to illustrate the effect of task partitioning on job workload for big data applications, which is the foundation of this research. Towards this goal, we implement a MapReduce program to find out the most common reason for flight cancellations based on the airline on-time performance data set from [1] and run this program on a computer server equipped with 2 processors of Intel(R) Xeon(R) CPU E5-2630 with 6 cores of 2.30GHz and 64GB memory. The program execution time is measured and plotted in Fig. 4, which shows that the execution time of this MapReduce job increases as the number of tasks increases when the server is fully utilized during execution, which means that the total workload increases with the number of tasks.

6.2 Simulation Settings

We 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 layers, 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 layer except the first and last layers, in which there is only one job; (iii) for each job, add an input edge from a randomly selected job in the immediately preceding layer, if absent, and an output edge to a randomly selected job in its downstream layer(s); (iv) randomly pick up two jobs in different layers and add a directed edge from the job in the upstream layer to the job in the downstream layer 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. 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 CPU-bound instructions of a task in each job on each type of machine is randomly selected between 0.1 and 1.0. The memory demand of a task in each job is randomly selected from 0.5GB to 4GB at an interval of 0.5GB.

Table 4: 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

We evaluate these algorithms in a heterogeneous cluster consisting of machines with four different specifications listed in Table 4, 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 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. The parameter ϵ in BAWMEE is set to 0.2. 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 0.1.

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 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.

6.3 Simulation Results

6.3.1 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 5. 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

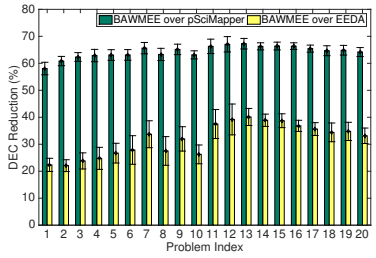

Figure 5: DECR vs. problem sizes.

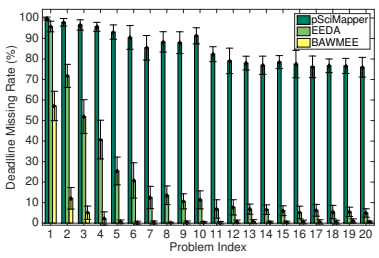

Figure 6: DMR vs. problem sizes.

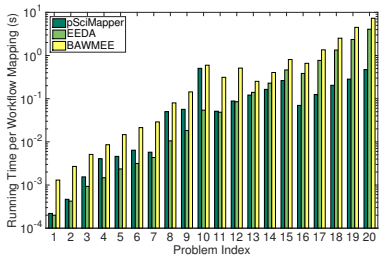

Figure 7: URT vs. problem sizes.

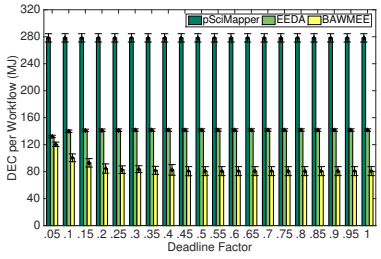

Figure 8: DEC vs. deadlines.

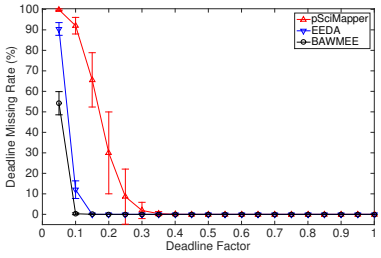

Figure 9: DMR vs. deadlines.

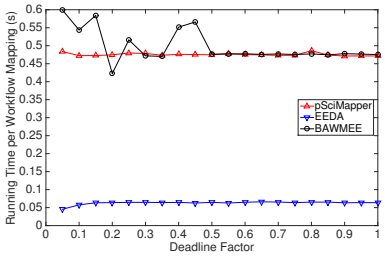

Figure 10: URT vs. deadlines.

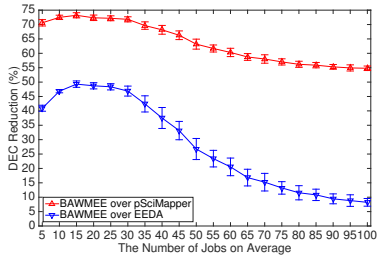

Figure 11: DECR vs. workflow sizes.

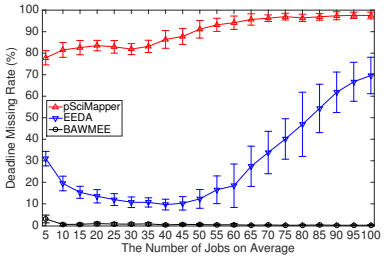

Figure 12: DMR vs. workflow sizes.

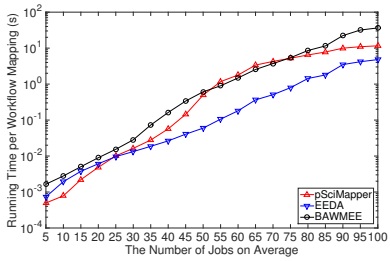

Figure 13: URT vs. workflow sizes.

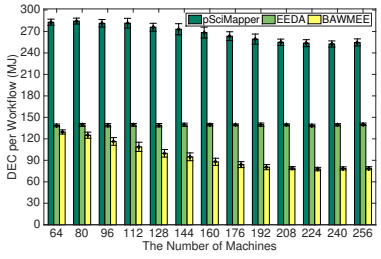

Figure 14: DEC vs. cluster sizes.

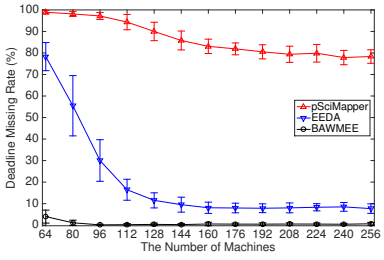

Figure 15: DMR vs. cluster sizes.

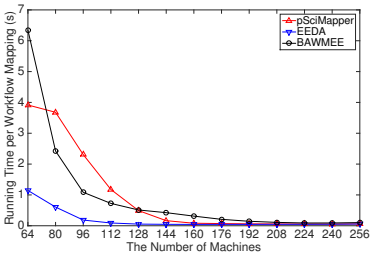

Figure 16: URT vs. cluster sizes.

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 0.1. We plot the DECR, DMR, and URT of pSciMapper, EEDA, and BAWMEE in Figs. 5-7, respectively, which show that BAWMEE saves 58.1% to 67.3% DEC and 22.1% to 40.1% DEC in comparison with pSciMapper and EEDA, respectively, and the DMR of pSciMapper (or EEDA) minus that of BAWMEE is in the range of 42.6% to 93.4% (or 4.1% to 59.7%). Furthermore, the URT of BAWMEE is on the same order of magnitude as those of pSciMapper and EEDA and is less than 7.4 seconds even for problem index 20.

6.3.2 *Deadline Constraint*

We evaluate the performance of pSciMapper, EEDA, 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. 8-10, respectively. These measurements

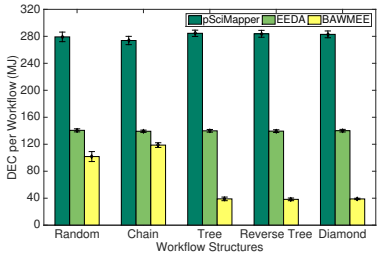

Figure 17: DEC vs. workflow structures.

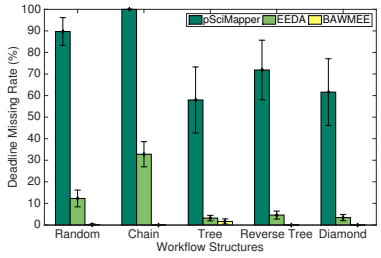

Figure 18: DMR vs. workflow structures.

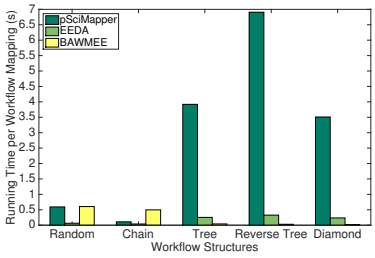

Figure 19: URT vs. workflow structures.

Table 5: 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)

show that BAWMEE saves 56.7% to 71.0% DEC and 8.8% to 43.1% DEC as the deadline increases, and reduces DMR from 99.8% and 90.4% to 54.2% with a deadline factor of 0.05 in comparison with pSciMapper and EEDA, respectively. The DMR of BAWMEE is close to zero when the deadline factor is larger than 0.1. Additionally, the URT of BAWMEE is less than 0.6 second and is 89.1% to 1.2 times and 6.6 to 13.2 times of those of pSciMapper and EEDA, 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. Hence, BAWMEE makes a better tradeoff between DEC and DMR than the other algorithms in comparison at an acceptable cost of running time.

6.3.3 *Workflow Size*

For scalability test, we run these three algorithms under different average workflow sizes with 5 to 100 jobs per workflow at a step 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. 11-13, respectively, where we observe that BAWMEE achieves an increasing DECR between 54.8% and 73.2% in comparison with pSciMapper, and between 8.2% and 49.3% in comparison with EEDA. Moreover, BAWMEE only misses less than 3.0% deadlines while pSciMapper and EEDA miss 77.9% to 97.6% and 9.6% to 69.6% deadlines, respectively. For large workflow sizes with 80 to 100 jobs per workflow that impose high resource demands, BAWMEE achieves a DECR between 8.2% and 11.5%, because it significantly reduces DMR (the first objective) from over 54.3% to less than 0.1%, in comparison with EEDA. The DMR of EEDA experiences a slump under the medium workflow sizes because a higher accuracy could be achieved on 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 75.5% to 3.6 times and 1.3 to 10.2 times of those of pSciMapper and EEDA, respectively.

6.3.4 *Cluster Size*

We run these three 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. 14-16, respectively, where we observe that as the number of machines increases, BAWMEE consumes 54.2% to 69.3% and 6.6% to 43.9% less DEC than pSciMapper and EEDA, respectively, hence exhibiting a satisfactory scalability property with respect to the cluster size. Furthermore, DAWMEE only misses 0.1% to 4.0% deadlines while pSciMapper and EEDA miss 77.9% to 98.9% and 7.7% to 78.3% deadlines, respectively. The increase in the cluster size results in a relatively looser deadline and a more flexible workflow mapping, as a result of which, the DMRs of these three algorithms drastically decrease, and BAWMEE has more chances to save energy. Moreover, the URT of BAWMEE is less than 6.4 seconds and is comparable with those of pSciMapper and EEDA.

6.3.5 *Workflow Structure*

We further investigate these three 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. 17-19, respectively, which show that BAWMEE reduces DEC by 63.6%, 56.6%, 86.3%, 86.5% and 86.2% as well as by 27.6%, 14.7%, 72.1%, 72.5% and 72.2% in comparison with pSciMapper and EEDA in random, chain, tree, reverse tree and diamond structured workflows, respectively. Further, BAWMEE saves less energy in chain-structured workflows than others, because the deadline baseline is set to be the tightest for this structure based on our deadline generation method. BAWMEE only misses 0.2%, 0%, 1.6%, 0% and 0% deadlines, while pSciMapper and EEDA miss 89.7%, 100%, 58.0%, 71.9% and 61.6% deadlines, and 12.3%, 32.8%, 3.2%, 4.6% and 3.4% deadlines in random, chain, tree, reverse tree and diamond structured workflows, respectively. Besides, the URT of BAWMEE is less than 0.6 second, and is 1.0 times, 4.5 times, 1.1%, 0.4% and 0.4%, as well as 9.4 times, 12.7 times, 16.5%, 9.1% and 6.4% of those of pSciMapper and EEDA in random, chain, tree, reverse tree and diamond structured workflows, respectively.

7 Conclusion

We investigated the property of moldable jobs and formulated a workflow mapping problem to minimize dynamic en-

ergy consumption under deadline and resource constraints. We designed an FPTAS for a special case with a pipeline-structured workflow on a homogeneous cluster, which was proved to be NP-complete, and a heuristic for a generalized problem with an arbitrary workflow on a heterogeneous cluster. The performance superiority of the proposed heuristic in terms of dynamic energy saving and deadline missing rate was illustrated by extensive simulation results in Hadoop/YARN in comparison with existing algorithms.

Our work reveals that the energy-efficient and deadline-aware mapping algorithms tailored to big data workflows could lead to significant energy savings and a higher level of Quality of Service. It is of our future interest to incorporate the proposed mapping algorithms into the existing workflow engines in the Hadoop ecosystem including Oozie [2] and Tez [3], and evaluate the performance of energy saving for real-life big data workflows.

8 Acknowledgments

This research is sponsored by National Science Foundation under Grant No. CNS-1560698 with New Jersey Institute of Technology.

9 References

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

[2] Apache Oozie. https://oozie.apache.org.

[3] Apache Tez. https://tez.apache.org.

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

[5] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proc. of ACM SoCC*, pages 217–228, Indianapolis, IN, USA, Jun 2010.

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

[7] 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.

[8] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In *Proc. of ACM EuroSys*, pages 43–56, Bern, Switzerland, Apr 2012.

[9] D. Cheng, P. Lama, C. Jiang, and X. Zhou. Towards energy efficiency in heterogeneous Hadoop clusters by adaptive task assignment. In *Proc. of IEEE ICDCS*, pages 359–368, Columbus, OH, USA, Jun-Jul 2015.

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

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

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

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

[14] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging green energy in data-processing frameworks. In *Proc. of ACM EuroSys*, pages 57–70, Bern, Switzerland, Apr 2012.

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

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

[17] Y. C. Lee, H. Han, A. Y. Zomaya, and M. Yousif. Resource-efficient workflow scheduling in clouds. *Elsevier Knowledge-Based Systems*, 80:153–162, 2015.

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

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

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

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

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

[23] 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.

[24] P. Sanders and J. Speck. Energy efficient frequency scaling and scheduling for malleable tasks. In *Proc. of Euro-Par*, pages 167–178, Rhodes Island, Greece, Aug 2012.

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

[26] L. Zhang, K. Li, C. Li, and K. Li. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Elsevier Info. Sci.*, in press.

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

[28] Q. Zhu, J. Zhu, and G. Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proc. of ACM/IEEE SC*, pages 1–12, New Orleans, LA, USA, Nov 2010.