Energy-efficient Dynamic Scheduling of Deadline-constrained MapReduce Workflows

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

***Abstract*—Big data workflows comprised of moldable parallel MapReduce programs running on a large number of processors have become a main consumer of energy at data centers. The degree of parallelism of each moldable job in such workflows has a significant impact on the energy efficiency of parallel computing systems, which remains largely unexplored. In this paper, we validate with experimental results the moldable parallel computing model where the dynamic energy consumption of a moldable job increases with the number of parallel tasks. Based on our validation, we construct rigorous cost models and formulate a dynamic scheduling problem of deadline-constrained MapReduce workflows to minimize energy consumption in Hadoop systems. We propose a semi-dynamic online scheduling algorithm based on adaptive task partitioning to reduce dynamic energy consumption while meeting performance requirements from a global perspective, and also design the corresponding system modules for algorithm implementation in Hadoop architecture. The performance superiority of the proposed algorithm in terms of dynamic energy saving and deadline violation is illustrated by extensive simulation results in Hadoop/YARN in comparison with existing algorithms, and the core module of adaptive task partitioning is further validated through real-life workflow implementation and experimental results using the Oozie workflow engine in Hadoop/YARN systems.**

I. Introduction

Big data analytics require the invocation and coordination of many computing tools, programs, libraries, services, or systems with complex execution dependencies, which are increasingly managed by workflow technologies. Big data workflows are typically comprised of moldable parallel MapReduce programs running on a large number of processors and have become a main consumer of energy at data centers. Most previous research efforts on green computing consider a batch of independent MapReduce jobs in Hadoop or traditional workflows comprised of serial programs, mainly using two types of techniques for energy saving: i) task consolidation to reduce static energy consumption (SEC) by turning off idle servers [6], [9], [3], [7], and ii) load balancing to reduce dynamic energy consumption (DEC) through dynamic voltage and frequency scaling (DVFS) [12], [15], [17], [26], [25], or a combination of both. However, these techniques are inadequate 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) there is no accurate model to describe the complicated relationship between DEC and CPU frequency, which poses a significant challenge to all DVFS-based scheduling.

Parallel jobs are generally categorized into three classes with flexibility from low to high: rigid jobs 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 [10]. A moldable job typically follows a computing performance model where the workload of each component task decreases while the total workload increases as the number of allotted processors increases [11].

In this paper, we validate with experimental results that the DEC of a MapReduce job also increases with the number of its parallel tasks. Based on this, we direct our efforts to workflow scheduling for dynamic energy saving by adaptively determining the degree of parallelism in each MapReduce job to reduce the workload overhead while meeting a given performance requirement. Our approach is orthogonal to the aforementioned two commonly used green computing techniques, and would add an additional level of energy efficiency to current computing platforms processing big data workflows.

In [22] and [23], we formulated an energy-efficient deadline-constrained static mapping problem for a single workflow comprised of moldable jobs, which has been proved to be strongly NP-hard. In this paper, we focus on the dynamic scheduling for a set of MapReduce workflows to minimize DEC under deadline and resource constraints in a cluster. The execution dynamics among multiple workflows make this problem even more challenging.

There is a trade-off between energy cost and execution time of each component job with multiple degrees of task partitioning granularity. Purely static scheduling as in [22] and [23] requires *a priori* knowledge of exact job execution cost and an accurate snapshot of available computing resources to schedule an individual MapReduce job in its entirety. Such a scheduling approach is not best suited for production high-performance computing systems, which are typically shared by a large number of users with high dynamics in resource use. However, a fully dynamic scheduling approach, benefiting resource allocation in a shared system such as [8] and [9], may lack a global perspective to balance the trade-off between energy cost and execution time of component jobs in each workflow. Therefore, we propose a semi-dynamic scheduling method consisting of three phases: i) Phase I: static mapping of each workflow on a virtual homogeneous cluster to determine

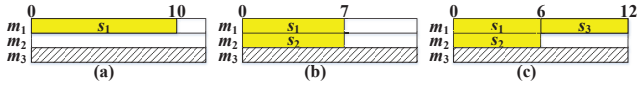

Fig. 1. A moldable job.

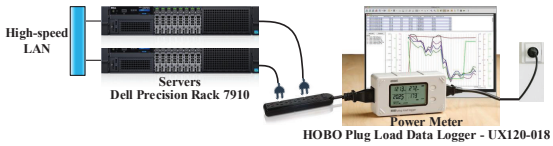

Fig. 2. The experimental testbed for measuring energy consumption [23].

the task partitioning of each component job, ii) Phase II: static mapping of each workflow on an idle heterogeneous cluster to set the virtual deadline of each component job, and iii) Phase III: dynamic resource allocation to ready-to-execute tasks based on their virtual job deadlines and the energy efficiency of heterogeneous machines. The first two static subproblems are processed by a workflow engine, such as Oozie and Tez, from the perspective of the entire workflow; the last dynamic subproblem is handled by the resource manager in Hadoop/YARN from the perspective of a shared system.

Our work makes the following contributions to the field.

- We validate with experimental measurements that the DEC of a MapReduce job in a Hadoop/YARN system increases with the number of parallel tasks, and analyze the performance variation.
- We propose a semi-dynamic online workflow scheduling algorithm, which adaptively reduces DEC from a global perspective in both temporal and spatial aspects, explicitly accounting for execution time estimation inaccuracies and computing system dynamics.
- The performance superiority of the proposed algorithm is illustrated by experimental results using the Oozie workflow engine in Hadoop/YARN systems and extensive simulation results in comparison with existing algorithms.

The rest of the paper is organized as follows. Section II validates the DEC model of a moldable job. Section III formulates a dynamic big data workflow scheduling problem. Section IV discusses the algorithm design principles. We design a heuristic and the corresponding system modules for algorithm implementation in Section V. Section VI presents performance evaluation. Section VII describes related work.

II. Performance Model of Moldable Jobs

In the performance model of a moldable job, as the number of component tasks increases, the workload of each task (which decides the task execution time) decreases, while the total workload of the job (which decides the job's DEC) increases. Hence, if the number of tasks does not exceed the maximum number of parallel tasks supported by the system, the job execution time is similar to the task execution time and thus decreases as the number of tasks increases; otherwise, both the job execution time and energy consumption may increase simultaneously with the number of tasks.

We present a numerical example in Fig. 1 to illustrate the possibility of reducing the execution time and energy consumption of a moldable job by properly adjusting the job's parallelism. In this example, there are three homogeneous machines, each of which can run at most one task, and two of which are idle at present. We consider a moldable job that can be partitioned into 1, 2, or 3 parallel tasks. In each of these three parallelization schemes, the workload of each task is 10G, 7G, and 6G flops, which take 10, 7, and 6 seconds to run and consume 10, 7, and 6 units of energy, respectively, and thus the total workload of the entire job is 10G, 14G, and 18G flops, which take 10, 7, and 12 seconds to run, and consume 10, 14, and 18 units of energy, respectively. Therefore, by changing the degree of parallelism in the job from 3 to 2, we are able to reduce the job execution time and DEC.

To validate this performance model, we conduct real-life experiments to illustrate the impact of the number of parallel tasks on DEC and execution time in big data computing applications, which lays down the foundation of this research.

A. 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 for each processor), 16GB 2133MHz DDR4 RDIMM ECC memory, and 256GB 2.5inch serial ATA solid state drive. We install a power meter of 0.5% measurement accuracy 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. 2. The initial measurement shows that the total static power consumption of this mini-cluster in an idle state is 153.5W on average.

On the cluster, we install Apache Hadoop 2.7.3 [4]. According to our Hadoop configuration, at most 23 map/reduce tasks in a MapReduce job can run on the cluster in parallel. We download the Wikipedia dataset from the PUMA website [2] and use the example MapReduce programs in Apache Hadoop 2.7.3 as standard benchmarks. We execute Grep on the 12GB/24GB dataset, referred to as Grep12/Grep24, and run WordCount on the 12GB dataset, referred to as WordCount12. We also download the airline on-time performance dataset of 11.2 GB for a period of 22 years (1987-2008) from the statistical computing website [1], and implement 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). To avoid block fragmentation in HDFS, we merge all the input data in a dataset into a single file, and then upload the merged file into HDFS.

We repeatedly run each MapReduce program with and/or without the reducing phase for 10 times on our cluster and measure the corresponding DEC and execution time of the mapping and reducing phases in each MapReduce job. To adjust the number of map-

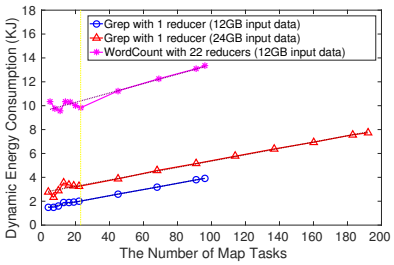

Fig. 3. Benchmarks: the DEC vs. the number of map tasks.

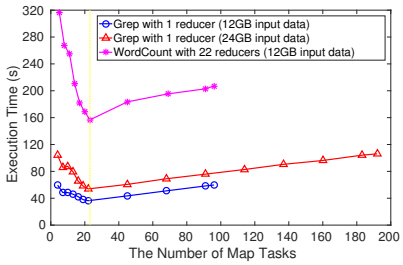

Fig. 4. Benchmarks: the execution time vs. the number of map tasks.

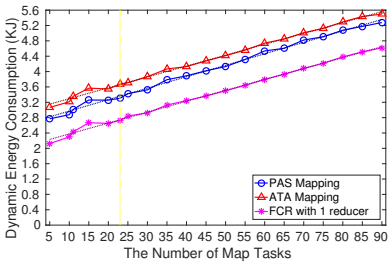

Fig. 5. The mapping phase of our statistical application: the DEC vs. the number of map tasks.

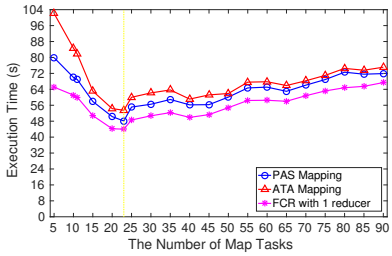

Fig. 6. The mapping phase of our statistical application: the execution time vs. the number of map tasks.

TABLE I
THE EXECUTION TIME AND DEC OF REDUCING VS. THE NUMBER OF REDUCERS.

The Number of Reducers	Reducing in PAS		Reducing in ATA	
	Time (s)	DEC (KJ)	Time (s)	DEC (KJ)
1	91.1	1.387	61.4	1.073
2	64.2	1.572	33.0	0.989
3	63.2	1.689	27.6	0.964
4	54.2	1.612	23.3	0.975
5	47.5	1.634	21.8	0.976
6	38.9	1.555	18.6	1.041
7	28.8	1.467	19.7	1.054
8			17.0	1.066

pers 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.

B. Experimental Results

We plot the average DEC and execution time of each MapReduce benchmark and the mapping phase of each statistical application in response to different numbers of map tasks in Figs. 3-6, respectively. These results show that the DEC of a MapReduce job increases with the number of map tasks, while its execution time decreases as the number of parallel map tasks in a single wave [13] increases up to 23, which is the largest number of map tasks supported simultaneously by the system. As the number of mappers increases from 5 (or 4) to 23 (or 22), the job execution time is only cut down by no more than a half (32.4% to 50.5%), because the concurrency of I/O operations for reading data is limited to 2 (i.e., the number of disks), and the time spent on the container initialization and the reducing phase is fixed. When the number of map tasks exceeds 23, the job execution time exhibits a fluctuant increase as the number of waves in the mapping phase increases from 2 to $\lceil 96/23 \rceil = 5$ for Grep12 and WordCount12, to $\lceil 192/23 \rceil = 9$ for Grep24, or to $\lceil 90/23 \rceil = 4$ for our applications. The fluctuation in Fig. 6 is due to the fact that the system capacity is not always fully utilized during the last wave of the mapping phase.

We conduct a linear fitting on the DEC measurements, and observe that the DEC of Jobs (including Grep12, Grep24, WordCount12, PAS, ATA, and FCR) with k mappers over their DEC with a single mapper follows a linear function with respect to k, i.e., $f_1(k) = 0.0181k + 0.9819$, $f_2(k) = 0.0097k + 0.9903$, $f_3(k) = 0.0041k + 0.9959$, $f_4(k) = 0.0108k + 0.965$, $f_5(k) = 0.0092k + 0.982$, and $f_6(k) = 0.0134k + 0.9835$, respectively, which justifies the parameter setting in Subsection VI-B1. The system log shows that there are 0 to 3 killed/resumed map tasks in each job execution, which explains the variations in Figs. 3 and 5.

We also tabulate the average DEC and execution time of the reducing phase in each MapReduce job in response to different numbers of tasks in Table I. Such a trend in reducing does not seem as obvious as that in mapping for two main reasons: i) There exists critical reduce-skew [14] for a small number of reduce keys. ii) Since the workload of reducing is much less than that of mapping in our MapReduce jobs, the measurement errors for reduce tasks in Table I are relatively larger in our measuring approach, where the DEC (or execution time) of reduce tasks is calculated as the difference between that of the entire job with 23 mappers and that of the corresponding map-only job with the same number of mappers.

III. Problem Formulation

A. Cost Models

1) Cluster Model

We consider a heterogeneous Hadoop cluster consisting of a set M of machines connected via high-speed switches, where each machine m_i is equipped with N_i homogeneous CPU cores of speed p_i and a shared memory of size o_i.

2) Workflow Model

We consider multiple user requests as a set of workflows $F = \{f_j(G_j, t_j, d_j)\}$, where f_j specifies a workflow structure

G_j, submission time t_j, and a deadline d_j. The structure of a workflow is defined as a directed acyclic graph (DAG) $G_j(V_j, A_j)$, where each vertex $v_{j,k} \in V_j$ represents a component job, and each directed edge $a_{j,k,k'} = (v_{j,k}, v_{j,k'}) \in A_j$ denotes an execution dependency. We treat the map and reduce phases in each job as two component jobs connected via a dependency edge. Each mapped job $v_{j,k}$ has its actual start time (AST) $t^S_{j,k}$ and actual finish time (AFT) $t^F_{j,k}$. We denote the completion time of an entire workflow f_j as t^C_j.

3) MapReduce Model

We consider a MapReduce job $v_{j,k}$ executing a set of parallel map (or reduce) tasks, each of which requires a memory of size $o_{j,k}$ and spends a percentage $\mu_{i,j,k}$ of time executing CPU-burst instructions on a CPU core of machine m_i. In job $v_{j,k}$, as the number $L_{j,k}$ of parallel tasks increases, the total workload $w_{j,k}(L_{j,k})$ of all tasks would increase and the workload $w_{j,k,l}(L_k) = w_{j,k}(L_{j,k})/L_{j,k}$ of each task $s_{j,k,l}$ would decrease. However, the maximum number $L'_{j,k}$ of tasks that can be executed in parallel without performance degradation is limited by the cluster capacity, e.g., $L'_k \leq \sum_{m_i \in M} \lfloor o_i/o_{j,k} \rfloor$. In addition, the execution time of task $s_{j,k,l}$ on machine m_i is $t_{i,j,k,l} = w_{j,k,l}(L_{j,k})/(\mu_{i,j,k} \cdot p_i)$. 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 [19], [21], which are beyond the scope of this work.

We denote the number of tasks in job $v_{j,k}$ mapped to machine m_i at time t as $n_{i,j,k}(t)$. The number of CPU cores and the amount of memory used by all component jobs in a set F of workflows on machine m_i at time t are $n_i(t) = \sum_{f_j \in F} \sum_{v_{j,k} \in V_j} n_{i,j,k}(t)$ and $o_i(t) = \sum_{f_j \in F} \sum_{v_{j,k} \in V_j} o_{j,k} n_{i,j,k}(t)$, respectively.

4) Energy Model

The DEC of a cluster is $E = \sum_{m_i \in M} \int_0^T P_i \sum_{f_j \in F} \sum_{v_{j,k} \in V_j} [\mu_{i,j,k} n_{i,j,k}(t)] dt$, 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 [8].

5) Mapping Function

We define a workflow mapping function as $\mathfrak{M} : \{s_l(v_k(f_j)) \xrightarrow{[t^S_{j,k,l},\ t^F_{j,k,l}]} m_i, \forall f_j \in F, \exists m_i \in M, \exists [t^S_{j,k,l}, t^F_{j,k,l}] \subset T\}$, which denotes that the l-th task of the k-th job of the j-th workflow is mapped to the i-th machine from time $t^S_{j,k,l}$ to time $t^F_{j,k,l}$. The domain of this mapping function covers all possible combinations of component jobs in a set F of workflows, a set M of machines, and a time period T of the cluster's operation [23].

B. Problem Definition

We formulate a dynamic energy-efficient workflow scheduling problem (EEWS) under deadline constraints:

Definition 1. ***EEWS****: Given a cluster of machines* $\{m_i(N_i, p_i, o_i, P_i)\}$, *and a set of workflows* $\{f_j(G_j(V_j, A_j), d_j)\}$ *whose arrivals follow Poisson distribution, where job* $v_{j,k}$ *in workflow* f_j *has a set* $\{w_{j,k}(L_{j,k}) | L_{j,k} = 1, 2, \ldots, L'_{j,k}\}$ *of workloads corresponding to different degrees of parallelism, and each task in job* $v_{j,k}$ *has a memory demand* $o_{j,k}$ *and spends a percentage* $\mu_{i,j,k}$ *of time executing CPU-burst instructions on machine* m_i*, we wish to find a mapping function* $\mathfrak{M} : (F, M, T) \rightarrow \{s_l(v_k(f_j)) \xrightarrow{[t^S_{j,k,l},\ t^E_{j,k,l}]} m_i\}$ *to minimize the dynamic energy consumption:*

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

subject to the following deadline, precedence, and resource constraints:

$$t^C_j \leq d_j, \forall f_j \in F$$
$$t^F_{j,k} \leq t^S_{j,k'}, \forall a_{j,k,k'} \in A_j, \forall f_j \in F$$
$$n_i(t) \leq N_i, \forall m_i \in M,$$
$$o_i(t) \leq o_i, \forall m_i \in M.$$

TABLE II
SCHEDULING IN A HETEROGENEOUS CLUSTER.

Algorithms	Parallelism	Scheduling	Decoupling	Long Tasks
BAWMEE	Yes	Purely static	No	Yes
EEDAW	No	Fully dynamic	No	No
MinD+ED	No	Semi-dynamic	Yes	Yes
ATP-EEDAW	Yes	Semi-dynamic	No	No
DAWSEE	Yes	Semi-dynamic	Yes	Yes

IV. THE DESIGN PRINCIPLES OF THE SCHEDULER

We first summarize the design of four algorithms adapted from different scenarios: i) BAWMEE in [23] that repeatedly maps each complete MapReduce workflow one at a time, ii) EEDAW adapted from a MapReduce job scheduling algorithm (integrated with the algorithms in [8] and [9]) by extending the progress estimation of a MapReduce job to that of a workflow, iii) MinD+ED adapted from a workflow scheduling algorithm with serial jobs in [27] by fixing the number of tasks in each MapReduce job and replacing preemptive task scheduling with non-preemptive task scheduling, and iv) ATP-EEDAW comprised of ATP proposed in Subsection V-B for static virtual mapping of each MapReduce workflow and EEDAW for dynamic energy-efficient and deadline-aware MapReduce workflow scheduling. However, these algorithms have their own weaknesses. BAWMEE, as a fully static scheduler, is not suited for systems shared by a large number of workflows; EEDAW, as a fully dynamic scheduler, lacks a global view to balance the tradeoff between energy cost and execution time of component jobs in each workflow; MinD+ED does not consider adaptive task partitioning for possible energy saving; ATP-EEDAW significantly increases the deadline missing rate in comparison to EEDAW because reducing the degree of parallelism for energy saving makes it more difficult to meet deadlines.

To overcome these weaknesses, we discuss three design principles of a scheduling algorithm for a set of MapReduce workflows in addition to adjusting the degree of parallelism in each component job, and tabulate the differences in Table II between the aforementioned four algorithms and our proposed method in Section V.

A. Dynamic Task Scheduling

Static mapping decides the mapping scheme for each entire workflow upon its arrival, while dynamic scheduling decides an on-demand mapping scheme for any ready-to-execute component of the workflows on the fly. Given a set of workflows, a greedy local optimization method such as BAWMEE in [23] repeatedly performs static mapping of each workflow. We adopt dynamic scheduling to consider resource sharing among multiple workflows from a global perspective.

B. Decoupling Dependencies and Shared Resources

In each workflow, the component jobs may affect each other's execution dynamics through the deadline and precedence constraints, while in a shared system, all ready-to-execute jobs may affect each other's execution dynamics through the resource constraint. Since both of these subproblems are NP-complete, conducting a joint optimization is very complicated. Setting an appropriate deadline for each component job in each workflow is an effective method to decouple the job dependencies in a workflow (i.e., temporal constraint) and resource sharing in the entire system (i.e., spatial constraint). As a result, the scheduler may only focus on the resource allocation for each job with its respective deadline, as each workflow is able to meet its deadline if all its jobs finish on time.

C. Avoiding Deadline Violation Caused by Heavyweight Tasks

Reducing the degree of parallelism increases the percentage of heavyweight tasks. In general, it is more challenging to schedule heavyweight tasks than lightweight ones to meet a certain deadline. In addition, in a heterogeneous cluster, the execution time and DEC of a task are unknown before it is assigned to a specific machine, so many existing research efforts consider an estimate based on the average or expected execution time and DEC [27]. The inaccuracy and uncertainty in such estimates further increase the difficulty of adjusting the workload of each component task in a moldable job. However, the heterogeneity of the cluster makes it possible to allocate more powerful computers to heavyweight tasks to meet the deadline constraint. Therefore, with inaccurate information, we may reduce the deadline missing rate if we consider the heterogeneity of machines after determining the degree of parallelism in each job.

V. Algorithm and System Design

In this section, we design a dynamic adaptive workflow scheduling algorithm for energy efficiency (DAWSEE).

A. DAWSEE Overview

The workflow scheduling process consists of three components: adaptive task partitioning (ATP), virtual deadline setting (VDS), and dynamic energy-efficient task scheduling (DEETS). Upon the arrival of a workflow request, ATP calculates the number of parallel tasks in each component job in the workflow according to the workflow's deadline, and each job's DEC (or workload) and average execution time across different numbers of parallel tasks on one machine across the entire cluster. Then, VDS computes an virtual deadline for each component job in the workflow, following the MinD algorithm in [27], whose main idea is to prioritize jobs with smaller dependencies on other jobs while extending their virtual deadlines based on the heterogeneity of machines, and to take equal slack time into account. We use MinD for VDS because MinD is able to balance the virtual deadlines of interdependent component jobs in a workflow and thus counteract the delay caused by ATP. Once a job is ready to execute, DEETS allows its tasks to wait for the most energy-efficient machines until its virtual deadline expires, or schedules them to machines to achieve the earliest finish time (EFT) if the former fails. Meanwhile, it prioritizes ready-to-execute jobs with tighter virtual deadlines in the process of resource allocation.

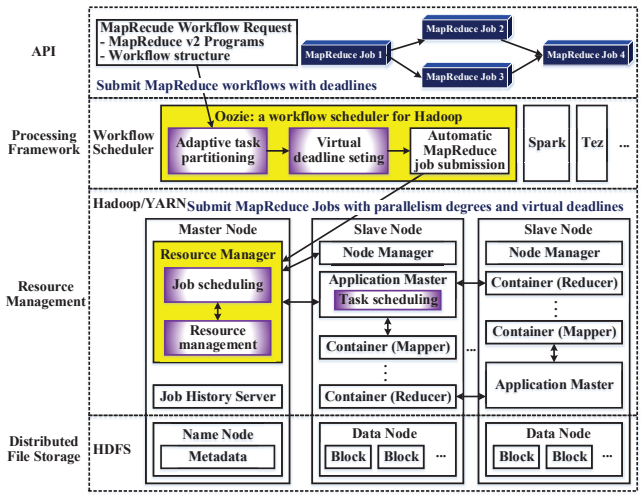

Fig. 7. The architecture of the MapReduce workflow scheduling system.

A data center typically has local storage attached to computing nodes, which are connected via high-speed switches. The workflow engine, such as Oozie [5], on Hadoop/YARN is responsible for handling MapReduce workflow requests. As shown in Fig. 7, we add two new modules, ATP and VDS, into Oozie. A submitted workflow is first processed by these two modules and its component jobs then wait for the completion of their preceding jobs in the automatic job submission module. Once a job is ready to execute, Oozie submits the job with its parallelism degree and virtual deadline to the resource manager in Hadoop/YARN, where the DEETS module sits. This architecture keeps the workflow engine and the Hadoop system loosely coupled and is best suited for our algorithm implementation.

B. Adaptive Task Partitioning

According to the system workload, we determine the current slack factor β, which is a coefficient multiplied by the difference between the deadline and the submission time of a workflow to decide its expected due as $d'_j = d_j - \beta \cdot (d_j - t_j)$.

Initially, we calculate the average task execution time (TET) and the average job DEC (JDEC) of each job across all possible numbers of parallel tasks running on one machine across the entire cluster (in Lines 1-4 of Alg. 1). Here, the

Algorithm 1: ATP()

Input: A workflow $f_j(G_j(V_j, A_j), t_j, d_j)$ and β

1: **for all** $v_{j,k} \in V_j$ **do**
2: $L'_{j,k} \leftarrow \min\{L'_{j,k}, \sum_{m_i \in M} \min\{N_i, \lfloor o_i/o_{j,k} \rfloor\}\}$;
3: **for** $L \leftarrow 1, \ldots, L'_{j,k}$ **do**
4: For job $v_{j,k}$ with L tasks, calculate its average TET $\tilde{t}_{j,k}(L)$ and average JDEC $\tilde{e}_{j,k}(L)$ on one machine across the entire cluster;
5: $t^{LF}_{j,k} \leftarrow +\infty$ for $\forall v_{j,k} \in f_j$;
6: $t^{LF}_{j,K} \leftarrow d'_j$ for the end job $v_{j,K}$ in f_j, where $d'_j = d_j - \beta \cdot (d_j - t_j)$;
7: Initialize unmapped workflow branches $G' \leftarrow G_j$;
8: **while** $G' \neq \emptyset$ **do**
9: Find the critical path cp ending at a job v_{j,k_1} with the earliest (T)LVFT in G' according to $\{\tilde{t}_{j,k}(1) | v_{j,k} \in G'\}$;
10: $cp.lvft \leftarrow t^{LF}_{j,k_1}$;
11: **if** $VPM(cp, \{(\tilde{t}_{j,k}(L), \tilde{e}_{j,k}(L)) | L \in [1, L'_{j,k}]\}) =$ False **then**
12: $v_{j,k_2} \leftarrow$ the last job with determined $L_{j,k}, t^{VS}_{j,k}$ and $t^{VF}_{j,k}$ in cp;
13: $D(v_{j,k_2}) \leftarrow$ {the downstream jobs of v_{j,k_2} in G_j};
14: **if** $\{v_{j,k'} \in D(v_{j,k_2}) | t^{VS}_{j,k'} < t^{VF}_{j,k_2}\} \neq \emptyset$ **then**
15: Clear $L_{j,k'}$, $t^{VS}_{j,k'}$ and $t^{VF}_{j,k'}$, and add $v_{j,k'}$ and its associated precedence constraints back to G';
16: $G' \leftarrow G' - \{v_{j,k} \in cp | L_{j,k}, t^{VS}_{j,k}$ and $t^{VF}_{j,k}$ are determined$\}$;

Algorithm 2: VPM()

Input: a pipeline pl with its EVST $pl.evst$ and LVFT $pl.lvft$ and a set of pairs $\{(\tilde{t}_{j,k}(L), \tilde{e}_{j,k}(L)) | v_{j,k} \in V_j, L \in [1, L'_{j,k}]\}$

Output: a boolean variable to indicate the nonexistence of precedence violation

1: Label the index k of each job in pl from 1 to the length of pl;
2: Update TEVST $t'_{ES}(v_{j,k})$ and TLVFT $t'_{LF}(v_{j,k})$ for $\forall v_{j,k} \in pl$;
3: **if** $\sum_{v_{j,k} \in pl} \tilde{t}_{j,k}(L'_{j,k}) < pl.lvft - pl.evst$ **then**
4: $t^{VS}_{j,1} \leftarrow pl.evst$; $t^{VF}_{j,1} \leftarrow t^{VS}_{j,1} + \tilde{t}_{j,1}(L'_{j,1})$; $L_{j,1} \leftarrow L'_{j,1}$;
5: **for** $v_{j,k} \in pl - \{v_{j,1}\}$ **do**
6: $t^{VS}_{j,k} \leftarrow \max\{t^{VF}_{j,k-1}, t'_{ES}(v_{j,k})\}$; $t^{VF}_{j,k} \leftarrow t^{VS}_{j,k} + \tilde{t}_{j,k}(L'_{j,k})$;
7: $L_{j,k} \leftarrow L'_{j,k}$;
8: **if** $t^{VF}_{j,k} > t'_{LF}(v_{j,k})$ **then**
9: **return** False;
10: **return** False.
11: Use Alg. 1 in [23] to calculate the number $L_{j,k}$ of tasks, VST $t^{VS}_{j,k}$ and VFT $t^{VF}_{j,k}$ for each job $v_{j,k}$ in pipeline pl, where each job $v_{j,k} \in pl$ has a set of pairs $\{(\tilde{t}_{j,k}(L), \tilde{e}_{j,k}(L))\}$;
12: **for** $v_{j,k+1} \in pl$ **do**
13: **if** $t^{VF}_{j,k} > t'_{LF}(v_{j,k})$ or $t^{VF}_{j,k} < t'_{ES}(v_{j,k+1})$ **then**
14: $pl(1,k).evst \leftarrow pl.evst$; // $pl(1,k)$ is a sub-pipeline from the first job to k-th job in pipeline pl
15: **if** $t^{VF}_{j,k} > t'_{LF}(v_{j,k})$ **then**
16: $pl(1,k).lvft \leftarrow t'_{LF}(v_{j,k})$;
17: **else**
18: $pl(1,k).lvft \leftarrow \min\{t'_{ES}(v_{j,k+1}), t'_{LF}(v_{j,k}), pl.lvft\}$;
19: Clear $L_{j,k}$, $t^{VS}_{j,k}$ and $t^{VF}_{j,k}$ for each job $v_{j,k}$ in pl;
20: **return** $VPM(pl(1,k), \{(\tilde{t}_{j,k}(L), \tilde{e}_{j,k}(L))\})$;
21: **return** True.

JDEC of $v_{j,k}$ with $L_{j,k}$ tasks on machine m_i can be computed as $e_{j,k}(L_{j,k}, m_i) = P_i w_{j,k}(L_{j,k})/p_i$. Based on the average TET and JDEC of each job, we are able to perform task partitioning and virtual mapping for each job upon workflow submission, i.e., to determine the number of tasks in each job and calculate its virtual start time (VST) $t^{VS}_{j,k}$ and virtual finish time (VFT) $t^{VF}_{j,k}$ without actually mapping the job to a specific machine. If all the preceding jobs of job $v_{j,k}$ are virtually mapped, its earliest virtual start time (EVST) $t^{ES}_{j,k}$ is the maximum VFT of its preceding jobs; if all the succeeding jobs of job $v_{j,k}$ are virtually mapped, its last virtual finish time (LVFT) $t^{LF}_{j,k}$ is the minimum VST of its succeeding jobs. The EVST of the start job is t_j and the LVFT of the end job is d'_j. If there exist virtually unmapped preceding and succeeding jobs of $v_{j,k}$, we calculate its temporary earliest virtual start time (TEVST) $t'_{ES}(v_{j,k})$ and temporary last virtual finish time (TLVFT) $t'_{LF}(v_{j,k})$ based only on its virtually mapped preceding and succeeding jobs, respectively.

Alg. 1 for ATP consists of two components: iterative critical path (CP) selection, and virtual pipeline mapping (VPM) including the task partitioning of each job in a pipeline. i) it starts with computing an initial CP, which is the longest execution path in a workflow, according to the average execution time of each job running in serial on one machine across the entire cluster, followed by the VPM process. Then, it iteratively computes a CP with the earliest LVFT from the remaining unmapped workflow branches based on the same average execution time of each job as above and performs the VPM of the computed CP until there are no branches left (in Lines 9 and 16). ii) For each selected CP, we perform the VPM for the CP by calling $VPM()$ in Alg. 2. If the pipeline has any job whose virtual mapping violates the precedence constraints, we cancel the virtual mapping of its downstream jobs whose VSTs are earlier than its VFT (in Lines 11-15). The virtual mapping of the first job on each previously selected CP would not be cancelled because the CP with the earliest LVFT is selected and virtually mapped in each iteration. Hence, Alg. 1 terminates after at most $|V_j|$ iterations.

In Alg. 2 for VPM, if a pipeline cannot meet its LVFT with each job $v_{j,k} \in pl$ divided into the maximum number $L'_{j,k}$ of tasks, VPM virtually maps each job $v_{j,k}$ with $L'_{j,k}$ tasks in their execution order until reaching a job that violates the precedence constraints, and then returns False (in Lines 3-10); otherwise, we consider the pipeline with its EVST and LVFT, where each job $v_{j,k}$ has a set of pairs of average TET and JDEC for different task partitioning, and use an FPTAS based on Alg. 1 in [23] to calculate the number of tasks and the virtual start and finish time for each job in the pipeline (in Line 11). Then, we check whether the VST and VFT of each job are between its TEVST and TLVFT in their execution order (in Lines 12-13). If there exists a job that violates the precedence constraints, we divide the pipeline at this job, and recursively call Alg. 2 to compute the virtual mapping of the upstream sub-pipeline with updated EVST and LVFT constraints (in Lines 13-20).

C. Virtual Deadline Setting

Initially, all jobs are assumed to run their fastest on their respective machines with the corresponding tightest deadlines. The priority of job v_k is set as $p_r(v_{j,k}) = -\sum_{v_{j,k'} \in R(v_{j,k})} w_{j,k'}(L_{j,k'})$, where $R(v_{j,k})$ is a set of jobs that have a path from/to $v_{j,k}$ in G_j. The job with the highest priority is considered to be virtually reassigned to a slower but more energy-efficient machine by one level for job deadline extension. Then, the rest of the jobs are considered in the

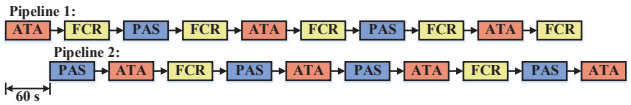

Fig. 8. Pipeline-structured MapReduce workflows.

order, followed by another round, if possible. As a result, it is more likely for the virtual deadlines of lightweight jobs to be extended than those of heavyweight jobs, which, to some extent, counteracts the delay caused by heavyweight jobs. Please refer to [27] about how MinD applies equal slack time to further job deadline extension in the next step.

VI. Performance Evaluation

We first conduct experiments to evaluate the performance of ATP in comparison with the default workflow scheduling schemes in Oozie and Hadoop systems. We then conduct simulations to evaluate the performance of DAWSEE in comparison with BAWMEE, EEDAW, MinD+ED, and ATP-EEDAW. In EEDAW and MinD+ED 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.

A. Experiments

1) Experimental Settings

The testbed is the same as described in Subsection II-A. On the cluster in our testbed, we also install Oozie 4.3 [5], a workflow engine that dispatches each component MapReduce job in a workflow with its respective configuration once all its preceding jobs finish. We generate two pipeline-structured workflows, each comprised of 10 MapReduce jobs, as shown in Fig. 8. These jobs are randomly selected from the aforementioned three MapReduce programs: PAS, ATA, and FCR. Here, we consider pipelines as this is one typical workflow structure supported by our cluster testbed.

Since EEDAW and MinD+ED do not adjust the number of mappers/reducers and only employ the heterogeneity of machines for energy saving, they produce identical scheduling schemes on a homogeneous cluster that strongly rely on the default settings in Hadoop, where the number of mappers is the input data size divided by the split size of 128 MB, and the number of reducers is 1. Hence, we refer to the mapping scheme produced by these two algorithms as the "default" scheme in this scenario.

2) Experimental Results

Although ATP can be treated as one preprocessing phase in MapReduce workflow scheduling, it is the most important component of DAWSEE to employ the property of Moldable jobs to save DEC. To test the practical energy saving achieved by the ATP component for multiple workflows, we conduct an experiment of scheduling these two pipeline-structured MapReduce workflows on our homogeneous cluster, where the second pipeline arrives 60 seconds after the arrival of the first one as shown in Fig. 8, and plot in Figs. 9 and 10 the analytical estimates and experimental measurements of the DEC and completion time based on the default workflow mapping scheme, as well as the scheme produced by ATP in various cases with different deadline constraints for each workflow, different slack factor β, and different approximate ratios ϵ in ATP. The horizontal axis represents different cases. For example, in the first case for ATP, the deadlines of the first and second pipelines are 960 seconds and 1165 seconds, respectively, and $\beta = 12\%, \epsilon = 0.2$. The experimental measurements show that ATP cuts down DEC by 34.5% to 38.8%, as well as completion time by 3.2% to 27.8% for the first pipeline and by 16.6% to 35.3% for the second pipeline in comparison with the default mapping scheme. These results clearly illustrate the dramatic dynamic energy saving and execution time reduction by ATP for multiple MapReduce workflows in practice. Due to the competition for shared resources, the workflow slack factors are set to be 12% to 26% in different cases to allow some extra time for avoiding missing deadline. However, the completion time of the first and second pipelines is in the same order as their deadlines, which shows that to some extent, ATP is able to balance the resource usage among multiple workflows according to their performance requirements. Furthermore, we observe that the differences between the analytical estimates and the experimental measurements of DEC are less than 2.0% for the mapping schemes produced by ATP, and 5.1% for the default mapping scheme, which indicates the accuracy of our cost models in describing the characteristics of workflow execution on a real Hadoop cluster.

To consider the parameter setting of ϵ in ATP, we conduct another workflow experiment of scheduling the first workflow on our cluster, and plot in Figs. 11 and 12 the analytical estimates and experimental measurements of the DEC and completion time based on the workflow mapping scheme produced by ATP, as well as the default and optimal workflow mapping schemes under 10 different deadline constraints without workflow slack time. The experimental measurements show that ATP with $\epsilon = 0.2$ cuts down DEC by 31.6% to 36.8% and completion time by at least 12.6% in comparison with the default mapping scheme, and consumes only 7.8% more dynamic energy in comparison with the optimal mapping scheme. Particularly, when $\epsilon = 0.2$, the completion time is less than the deadline over all the cases. The experimental results from the second pipeline are qualitatively similar. Therefore, we may set ϵ to be 0.2 to limit the computing time of mapping schemes in practice, and use it in the following simulation.

B. Simulation

1) Simulation Settings

We generate a set of random workflows using the method in Subsection 6.3 of [23]. 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 30 and 120. The workload of a job is randomly selected between 0.6×10^{12} and 21.6×10^{12} CPU cycles when running in serial. Based on our measurements in Subsection II-B, the workload $w(k)$

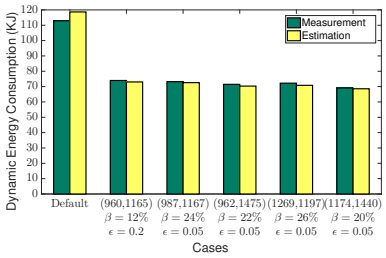

Fig. 9. The DEC of a pair of pipelines with different approximate ratios under different deadline constraints.

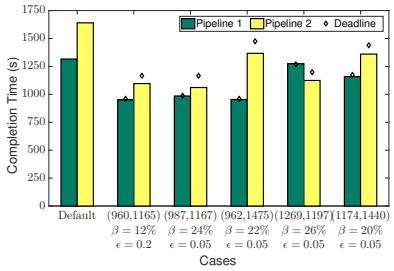

Fig. 10. The completion time of a pair of pipelines with different approximate ratios under different deadline constraints.

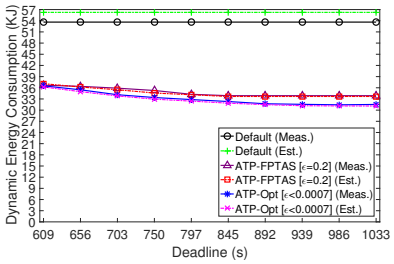

Fig. 11. The DEC of Pipeline 1 under different deadline constraints.

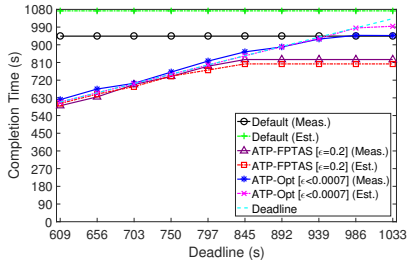

Fig. 12. The completion time of Pipeline 1 under different deadline constraints.

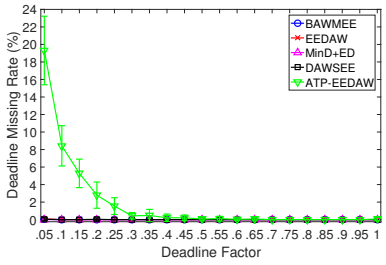

Fig. 13. The DMR vs. deadlines.

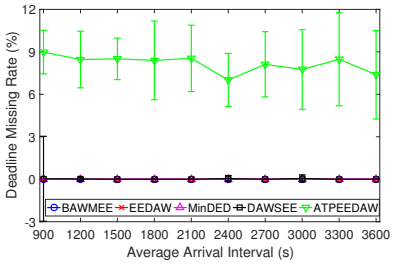

Fig. 14. The DMR vs. arrival intervals.

of a job with $k > 1$ tasks is randomly selected between $w(1)[1 + \alpha \cdot (k - 1.4)]$ and $w(1)[1 + \alpha \cdot (k - 0.6)]$, where α is fixed for each job, but is randomly selected from the range of [0.009, 0.013] for different jobs. We calculate the sum of the average execution time of the serial jobs on the CP and set it as a deadline baseline. The percentage of execution time for CPU-burst instructions of a task in each job on each type of machine is randomly selected from 0.8, 0.9, and 1. The memory demand of a task in each job is randomly selected from 0.5GB to 4GB at an interval of 0.5GB.

We evaluate these algorithms in a heterogeneous cluster consisting of machines with four different specifications in Table 7 of [23], based on 4 types of Intel processors: 1) Six-core Xeon E7450, 2) Single Core Xeon, 3) Dual Core Xeon 7150N, and 4) Itanium 2 9152M. Each homogeneous sub-cluster has the same number of machines. Each simulation lasts for 3 days and is repeated 20 times with different workflow instances, whose arrivals follow Poisson distribution. In the evaluation, each data point represents the average of 20 runs with a standard deviation. The parameter ε in BAWMEE and DAWSEE is set to 0.2. By default, the workflow size is randomly selected from 40 to 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 decide the actual workflow deadline, is set to 0.1.

We define the DEC reduction (DECR) over the other algorithms in comparison as

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

where DEC_{DAWSEE} and DEC_{Other} are the average DEC per workflow achieved by DAWSEE 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.

2) Deadline Missing Rate

We evaluate the DMR of BAWMEE, EEDAW, MinD+ED, ATP-EEDAW, and DAWSEE with different deadline constraints, average workflow arrival intervals, average workflow sizes, and cluster sizes, and plot the DMR in Figs. 13-16, respectively. We observe that the DMR of all the algorithms except ATP-EEDAW is close to zero. The performance superiority of DAWSEE over ATP-EEDAW indicates that setting an appropriate job deadline and selecting a suitable machine for each job according to the job deadline would help reduce the execution time, which may have been prolonged by a lower degree of parallelism. Since the deadline requirement is of the highest priority, we do not compare ATP-EEDAW with others in terms of DEC in the rest of the simulation.

3) Dynamic Energy Saving

We evaluate the DEC of BAWMEE, EEDAW, MinD+ED, and DAWSEE 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 measurements of these algorithms are plotted in Fig. 17, which shows that DAWSEE saves DEC by 12.6% to 35.0%, 15.8% to 32.2%, and 30.4% to 41.8% as the deadline increases in comparison with BAWMEE, EEDAW, and MinD+ED, respectively. It is worth pointing out that as the deadline increases, the DEC of DAWSEE decreases due to a lower degree of parallelism. Furthermore, DAWSEE reduces the number of tasks much more significantly than BAWMEE because the ATP in DAWSEE ignores resource availability considered by subsequent DEETS.

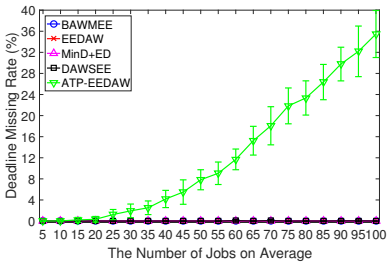

Fig. 15. The DMR vs. workflow sizes.

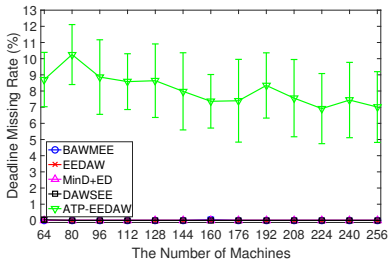

Fig. 16. The DMR vs. cluster sizes.

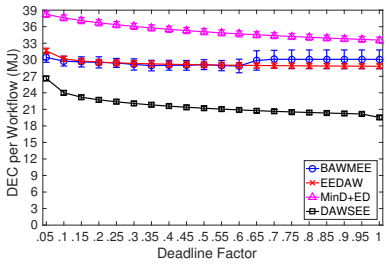

Fig. 17. The DEC vs. deadlines.

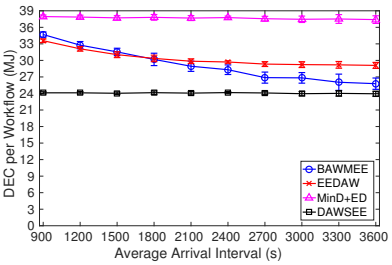

Fig. 18. The DEC vs. arrival intervals.

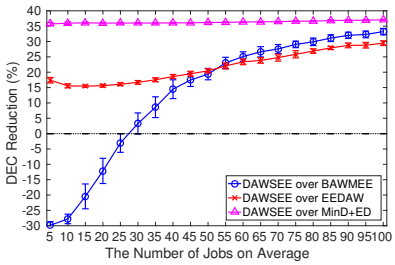

Fig. 19. The DECR vs. workflow sizes.

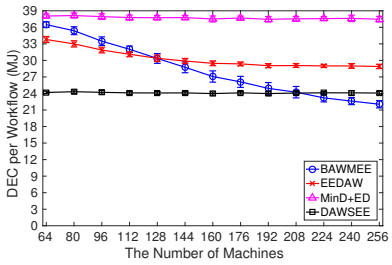

Fig. 20. The DEC vs. cluster sizes.

To evaluate dynamic adoption, we run these four algorithms under different average workflow arrival intervals of 15 to 60 minutes at a step of 5 minutes. The DEC measurements of these algorithms are plotted in Fig. 18, where we observe that as the arrival interval increases, DAWSEE consumes relatively stable DEC, which is 7.1% to 30.5%, 17.7% to 28.2%, and 35.9% to 36.5% less than the DEC of BAWMEE, EEDAW, and MinD+ED, respectively.

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 jobs. The maximum and minimum workflow sizes are 2 jobs more and less than the average workflow size, respectively. We plot the DECR of these algorithms in Fig. 19, where we observe that DAWSEE achieves an increased DECR from 15.5% to 29.5% and from 35.7% to 37.1% in comparison with EEDAW and MinD+ED, respectively. For small workflows with 5 to 25 jobs that demand less resources, DAWSEE achieves negative DECR over BAWMEE, because shared systems without resource competition lead to exclusive resource use and thus joint optimization of task partitioning and resource allocation in BAWMEE outperforms the respective optimization of separate subproblems in DAWSEE. However, the absolute DEC of small workflows is much less than that of large ones.

We run these four algorithms under different cluster sizes of 64 to 256 machines at a step of 16 machines for scalability test. The DEC measurements of these algorithms are plotted in Fig. 20, which shows that as the number of machines increases, DAWSEE consumes relatively fixed dynamic energy, which is 16.7% to 28.4% and 35.8% to 36.4% less than the DEC of EEDAW and MinD+ED, respectively, hence exhibiting a satisfactory scalability property with respect to the cluster size. As the cluster size scales up to 224 machines and beyond, the marginal performance gain of BAWMEE over DAWSEE in terms of DEC is not caused by the difference in the number of tasks. The reason is that with sufficient resources, the energy saving from balancing the resource use among workflows in DAWSEE is less than that from the joint optimization of task partitioning and resource allocation within a single workflow in BAWMEE. However, in such cases with sufficient resources, a more effective method for energy saving would be to selectively turn off some servers for SEC reduction.

VII. Related Work

A. Energy-efficient Dynamic Scheduling in Hadoop Clusters

Static MapReduce job scheduling for energy efficiency has been investigated in various contexts [6], [18]. We provide a survey on energy-efficient dynamic scheduling in Hadoop.

In large-scale clusters, servers are typically upgraded and replaced in an incremental manner. As a result, many techniques make use of the hardware heterogeneity of Hadoop clusters for energy saving. 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 heterogeneous MapReduce applications in a heterogeneous Hadoop cluster [8]. 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.

The majority of existing efforts on MapReduce job scheduling were devoted to Hadoop 1. The work on Hadoop 2, or YARN, is still quite limited. Li *et al.* proposed dynamic suspend-resume mechanisms to mitigate the overhead of preemption in cluster scheduling, and used a check pointing

mechanism to save the states of jobs for resumption [16]. As opposed to preemptive scheduling, our work is focused on energy saving in non-preemptive scheduling in the background.

B. Energy-efficient Dynamic Scheduling in Workflow Engines

Many efforts have been made on energy-efficient static workflow scheduling [15], [17], [24], but the work on dynamic scheduling is still quite limited. 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 [27]. Their work only considers serial or rigid jobs in workflows, while ours is focused on moldable jobs in big data systems.

C. Energy-efficient Malleable Job Scheduling

There exist relatively limited efforts on 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 [20]. Different from these theoretical models, our work employs measurement-based power consumption models and performs dynamic workflow scheduling 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 dynamic scheduling of big data workflows comprised of moldable jobs in Hadoop.

VIII. Conclusion

We investigated the properties of moldable MapReduce jobs and validated with experimental results that the DEC of a MapReduce job increases with the number of parallel tasks. Then, we formulated a dynamic scheduling problem of big data workflows to minimize energy consumption under deadline constraints in Hadoop systems with time-varying computing resources. To solve the problem, we designed a semi-dynamic online scheduling algorithm with adaptive task partitioning to reduce dynamic energy consumption while meeting performance requirements from a global perspective. The performance superiority of the proposed algorithm in term of dynamic energy saving and deadline miss rates is illustrated by extensive simulation results and further validated through real-life workflow implementation and experimental results using the Oozie workflow engine in Hadoop/YARN.

Acknowledgment

This research is sponsored by U.S. National Science Foundation under Grant No. CNS-1560698 with New Jersey Institute of Technology and National Nature Science Foundation of China under Grant No. 61472320 and U1609202 with Northwest University, P.R. China..

References

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

[2] 2017. PUMA Benchmarks and dataset downloads. https://engineering.purdue.edu/∼puma/datasets.htm.

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

[4] Apache, 2016. Hadoop. http://hadoop.apache.org.

[5] Apache, 2016. Oozie. https://oozie.apache.org.

[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] 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.

[8] 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.

[9] 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.

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

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

[12] S. Ibrahim, T.-D. Phanb, A. Carpen-Amarie, H.-E. Chihoubd, D. Moisee, and G. Antoniu. Governing energy consumption in Hadoop through CPU frequency scaling: An analysis. *Elsevier FGCS*, 54:219–232, 2016.

[13] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang. Hadoop performance modeling for job estimation and resource provisioning. *IEEE TPDS*, 27(2):441–454, 2016.

[14] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. SkewTune: Mitigating skew in mapreduce applications. In *Proc. of ACM SIGMOD*, pages 25–36, Scottsdale, AZ, USA, May 2012.

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

[16] 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.

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

[18] L. Mashayekhy, 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.

[19] F. A. S. Verboven, P. Hellinckx and J. Broeckhove. Runtime prediction based grid scheduling of parameter sweep jobs. In *Proc. of IEEE Asia-Pacific Services Computing Conference*, pages 33–38, Taiwan, Dec 2008.

[20] 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.

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

[22] T. Shu and 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*, pages 34–43, Salt Lake City, UT, USA, Nov 2016.

[23] T. Shu and C. Q. Wu. Energy-efficient mapping of large-scale workflows under deadline constraints in big data computing systems. *Elsevier FGCS*, 2017.

[24] 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.

[25] 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.*, 379:241–256, 2017.

[26] 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.

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