New Jersey Institute of Technology
Digital Commons @ NJIT

Dissertations | Theses and Dissertations

Summer 2017

Performance optimization and energy efficiency of big-data computing workflows

Tong Shu
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/dissertations

 Part of the Computer Sciences Commons

Recommended Citation
Shu, Tong, "Performance optimization and energy efficiency of big-data computing workflows" (2017). *Dissertations*. 41.
https://digitalcommons.njit.edu/dissertations/41

This Dissertation is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Dissertations by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

PERFORMANCE OPTIMIZATION AND ENERGY EFFICIENCY OF BIG-DATA COMPUTING WORKFLOWS

by
Tong Shu

Next-generation e-science is producing colossal amounts of data, now frequently termed as Big Data, on the order of terabyte at present and petabyte or even exabyte in the predictable future. These scientific applications typically feature data-intensive workflows comprised of moldable parallel computing jobs, such as MapReduce, with intricate inter-job dependencies. The granularity of task partitioning in each moldable job of such big data workflows has a significant impact on workflow completion time, energy consumption, and financial cost if executed in clouds, which remains largely unexplored. This dissertation conducts an in-depth investigation into the properties of moldable jobs and provides an experiment-based validation of the performance model where the total workload of a moldable job increases along with the degree of parallelism. Furthermore, this dissertation conducts rigorous research on workflow execution dynamics in resource sharing environments and explores the interactions between workflow mapping and task scheduling on various computing platforms. A workflow optimization architecture is developed to seamlessly integrate three interrelated technical components, i.e., resource allocation, job mapping, and task scheduling.

Cloud computing provides a cost-effective computing platform for big data workflows where moldable parallel computing models are widely applied to meet stringent performance requirements. Based on the moldable parallel computing performance model, a big-data workflow mapping model is constructed and a workflow mapping problem is formulated to minimize workflow makespan under a budget constraint in public clouds. This dissertation shows this problem to be strongly

NP-complete and designs i) a fully polynomial-time approximation scheme for a special case with a pipeline-structured workflow executed on virtual machines of a single class, and ii) a heuristic for a generalized problem with an arbitrary directed acyclic graph-structured workflow executed on virtual machines of multiple classes. The performance superiority of the proposed solution is illustrated by extensive simulation-based results in Hadoop/YARN in comparison with existing workflow mapping models and algorithms.

Considering that large-scale workflows for big data analytics have become a main consumer of energy in data centers, this dissertation also delves into the problem of static workflow mapping to minimize the dynamic energy consumption of a workflow request under a deadline constraint in Hadoop clusters, which is shown to be strongly NP-hard. A fully polynomial-time approximation scheme is designed for a special case with a pipeline-structured workflow on a homogeneous cluster and a heuristic is designed for the generalized problem with an arbitrary directed acyclic graph-structured workflow on a heterogeneous cluster. This problem is further extended to a dynamic version with deadline-constrained MapReduce workflows to minimize dynamic energy consumption in Hadoop clusters. This dissertation proposes 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 develops corresponding system modules for algorithm implementation in the Hadoop ecosystem. The performance superiority of the proposed solutions in terms of dynamic energy saving and deadline missing rate is illustrated by extensive simulation results in comparison with existing algorithms, and further validated through real-life workflow implementation and experiments using the Oozie workflow engine in Hadoop/YARN systems.

PERFORMANCE OPTIMIZATION AND ENERGY EFFICIENCY OF BIG-DATA COMPUTING WORKFLOWS

by
Tong Shu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2017

Copyright © 2017 by Tong Shu
ALL RIGHTS RESERVED

APPROVAL PAGE

PERFORMANCE OPTIMIZATION AND ENERGY EFFICIENCY OF BIG-DATA COMPUTING WORKFLOWS

Tong Shu

Dr. Chase Q. Wu, Dissertation Advisor — Date
Associate Professor of Computer Science, NJIT

Dr. Guiling Wang, Committee Member — Date
Professor of Computer Science, NJIT

Dr. Roberto Rojas-Cessa, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Andrew Sohn, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Xiaoning Ding, Committee Member — Date
Assistant Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Tong Shu

Degree: Doctor of Philosophy

Date: August 2017

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2017
- Master of Science in Computer Science,
  University of Memphis, Memphis, TN, 2015
- Bachelor of Science in Information Management and System,
  Peking University, Beijing, P.R. China, 2005

Major: Computer Science

Presentations and Publications:

Tong Shu and Chase Q. Wu. "Energy-efficient Dynamic Scheduling of Deadline-constrained MapReduce Workflows". *Submitted to the 13th IEEE International Conference on eScience*, 10 pages, Auckland, New Zealand, Oct 24th-27th 2017.

Tong Shu and Chase Q. Wu. "Energy-efficient Mapping of Large-scale Workflows under Deadline Constraints in Big Data Computing Systems". *Elsevier Future Generation Computing Systems*, 16 pages. (Impact factor: 3.997)

Tong Shu and Chase Q. Wu. "Bandwidth Scheduling for Energy Efficiency in High-performance Networks". *IEEE Transactions on Communications*, 15 pages. (Impact factor: 4.058)

Tong Shu and Chase Q. Wu. "Performance Optimization of Hadoop Workflows in Public Clouds through Adaptive Task Partitioning". *Proceedings of the 36th IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2349-2357, Atlanta, GA, USA, May 1st-4th 2017. (Acceptance rate: 20.9%)

Tong Shu and Chase Q. Wu. "Energy-efficient Mapping of Big Data Workflows under Deadline Constraints". *Proceedings of the 11th Workshop on Workflows in Support of Large-Scale Science (WORKS) in conjunction with ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 34-43, Salt Lake City, UT, USA, Nov 14th 2016. (Acceptance rate: 29.4%)

Tong Shu, Chase Q. Wu, and Daqing Yun. "Advance Bandwidth Reservation for Energy Efficiency in High-performance Networks". *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*, pp. 541-548, Sydney, Australia, Oct 21st-24th 2013. (Acceptance rate: 26.4%)

Tong Shu, Min Liu, Zhongcheng Li, and Chase Q. Wu. "Interference Pair-Based Distributed Spectrum Allocation in Wireless Mesh Networks with Frequency-Agile Radios". *Proceedings of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 82-90, Salt Lake City, UT, USA, Jun 27th-30th 2011. (Acceptance rate: 22%)

Tong Shu, Min Liu, and Zhongcheng Li. "Spectrum Allocation for Distributed Throughput Maximization under Secondary Interference Constraints in Wireless Mesh Networks". *Proceedings of the 20th IEEE International Conference on Computer Communications and Networks (ICCCN)*, 6 pages, Maui, HI, USA, Jul 31st-Aug 4th 2011. (Acceptance rate: 29.6%)

Anfu Zhou, Min Liu, Tong Shu, Yilin Song, and Zhongcheng Li. "Exploiting the Full Potential of Multi-AP Diversity in Centralized WLANs through Back-pressure Scheduling". *Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN)*, pp. 505-513, Bonn, Germany, Oct 4th-7th 2011. (Acceptance rate: 29.3%)

Tong Shu, Min Liu, Zhongcheng Li, and Anfu Zhou. "A Diagnosis-Based Soft Vertical Handoff Mechanism for TCP Performance Improvement". *Proceedings of the 19th IEEE International Conference on Computer Communications and Networks (ICCCN)*, 6 pages, Zurich, Switzerland, Aug 2nd-5th 2010. (Acceptance rate: 33.9%)

Tong Shu, Min Liu, Zhongcheng Li, and Anfu Zhou. "Joint Variable Width Spectrum Allocation and Link Scheduling for Wireless Mesh Networks". *Proceedings of IEEE International Conference on Communications (ICC)*, 5 pages, Cape Town, South Africa, May 23rd-27th 2010. (Acceptance rate: 39.5%)

Tong Shu, Min Liu, and Zhongcheng Li. "A Performance Evaluation Model for RSS-based Vertical Handoff Algorithms". *Proceedings of the 14th IEEE Symposium on Computers and Communications (ISCC)*, pp. 271-276, Sousse, Tunisia, Jul 5th-8th 2009. (Acceptance rate: 35.9%)

Tong Shu, Min Liu, Zhongcheng Li, and K. Zheng. "Network-layer Soft Vertical Handoff Schemes without Packet Reordering (short paper)". *Proceedings of the 34th IEEE Conference on Local Computer Networks (LCN)*, pp. 285-288, Zurich, Switzerland, Oct 20th-23rd 2009.

Dedicated to my beloved parents:
Licheng Shu *and* Baozhu Liu

ACKNOWLEDGMENT

I would like to express my deepest appreciation to my research advisor, Dr. Chase Wu, who provided me with valuable technical and writing guidance during my Ph.D. study. In the past five years, he has guided me through my application to the Ph.D. program, settle-down in the United States, and preparation for job hunting. In countless occasions, I have witnessed his great passion for excellence in research, his extreme diligence at work, and his immense patience with students. His extensive knowledge of life and strong expertise in research have had a profound influence on me, which will be of true value in my future career.

I also want to express my sincere gratitude to the members of my dissertation committee, including Dr. Xiaoning Ding, Dr. Andrew Sohn, Dr. Roberto Rojas-Cessa, and Dr. Guiling Wang, each of whom provided me with useful guidance, timely feedback, and constructive comments on my dissertation, which have greatly helped me improve my research in many technical aspects.

I am deeply thankful to all the faulty and staff members in the Department of Computer Science at New Jersey Institute of Technology and the University of Memphis. I would particularly like to thank Dr. Usman W. Roshan, Dr. James M. Calvin, Dr. Jason T. Wang, etc., for allowing me to audit their classes. The knowledge I learned from their classes is very important to the successful completion of my dissertation.

I would like to thank my parents for their constant love, support, and encouragement, and my English tutors, Dr. Jerome Paris, Dr. Janet M. Bodner, Ms. Vickey Smith, Mr. Kent Smith, and Mr. Wesley Bailey, for their help with the improvement of my English.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Computation-based analyses and simulations have become an essential research and discovery tool in next-generation scientific applications and are producing colossal amounts of data, now frequently termed as "Big Data", on the order of terabyte at present and petabyte or even exabyte in the predictable future. Other scientific data of similar scales generated in broad science communities include environmental observation data (satellite climate data [4, 5], multimodal sensor data, etc.), experimental measurement data (Spallation Neutron Source [6], Large Hadron Collider [7], etc.), and astronomical image data (Dark Energy Camera [29], Large Synoptic Sky Survey [59], etc.). Such datasets are increasingly managed and processed by scientific workflows of different structures as simple as a pipeline or as complex as a directed acyclic graph (DAG), which are typically executed in public clouds or local cluster computing environments.

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in data centers providing those services [14]. Public cloud platforms, such as Amazon EC2, Microsoft Azure, and Google Cloud, provide a virtual computing service as utility to meet time-varying computing demands. Within cloud platforms, scientific computing no longer requires large capital outlays in hardware purchases to deploy services or human expenses for operations. Moreover, public clouds provide various combinations of computing resources at different scales to meet disparate computing demands in different application domains. This elasticity of resources is attractive to many domain experts who intend to maximize their research outcome under limited budget. While reaping the benefits of cloud computing, scientific users are also facing two emerging

challenges, i.e., the feasibility and performance of computing services in clouds. The former is to minimize the financial expense of the entire workflow while the latter is to minimize the makespan of the entire workflow under a given budget constraint. In this dissertation, we consider two approaches, i.e., i) reduce the required budget by applying cost-aware job mapping models and ii) reduce the workflow makespan under a given budget constraint by designing cost-effective workflow mapping algorithms.

However, security and privacy are still a pressing issue of using public clouds as the data and operations must be sent to a third-party cloud service provider. Many efforts have been made in this regard. For example, homomorphic encryption is a form of encryption that allows computations to be carried out on ciphertext, thus generating an encrypted result, which, when decrypted, matches the result of operations performed on the plaintext. However, homomorphic encryption can only be applied to a limited set of computing operations. So far, physical separation still provides the highest-level security and datasets of certain confidentiality are oftentimes stored and processed in private clusters, which consume a significant amount of energy on a daily basis. Hence, in this dissertation, we also consider the reduction of energy consumption of big-data computing workflows in a shared cluster composed of physical machines (PMs) under deadline constraints.

Parallel jobs are generally categorized into three classes with flexibility from low to high: i) rigid jobs exemplified by multi-threaded programs running on a fixed number of processors, ii) moldable jobs exemplified by MapReduce programs running on any number of processors decided prior to execution, and iii) malleable jobs running on a variable number of processors at runtime [24]. Most existing efforts on workflow optimization in terms of makespan and energy efficiency consider serial or rigid jobs with execution dependencies, and a few efforts have been made to minimize the completion time of a workflow comprised of malleable jobs. However, there exist relatively limited efforts on moldable job mapping/scheduling for workflow

performance optimization and energy efficiency. In big data systems such as Hadoop/Spark, there arises a new challenge to optimize the mapping/scheduling of moldable parallel jobs, each of which has a variable number of independent homogenous tasks. In this dissertation, we focus on the performance optimization and energy efficiency of scientific workflows composed of moldable parallel jobs.

Furthermore, the end-to-end performance and energy efficiency of such scientific workflows depend on both the mapping scheme, which determines task assignment, and the scheduling policy, which determines resource allocation if multiple tasks are mapped to the same node. These two aspects of a workflow-based research process are traditionally treated as two separate topics, and the interactions between them have not been fully explored by any existing efforts. As the scale and complexity of scientific workflows environments rapidly increase, each individual aspect of performance optimization alone can only meet with limited success.

The main goal of this dissertation is to analyze and optimize the performance and energy consumption of large-scale scientific workflows by conducting an in-depth investigation into the property of moldable parallel jobs and exploring the interactions between workflow mapping and task scheduling. Towards this goal, we propose to build a layered workflow architecture that seamlessly integrates three interrelated technical components, i.e., resource allocation, job mapping, and task scheduling, based on rigorous algorithmic design, theoretical dynamics analysis, and real system implementation and evaluation. Within this architecture, we (i) construct rigorous cost models to describe the characteristics of real Hadoop/YARN systems in data centers, (ii) formulate a class of optimization problems for scientific workflows comprised of moldable jobs to minimize the end-to-end delay and feasible budget in public clouds, and minimize the energy consumption and deadline missing rate in shared clusters; (iii) design a set of cooperative workflow mapping and scheduling algorithms with mathematically proved optimality or correctness to cope with both

resource and user dynamics and achieve optimal workflow performances, (iv) develop a formal simulation program to validate cost models and theoretical results, and (v) implement, deploy, and execute real-life scientific workflows to demonstrate the performance superiority of the proposed algorithms in practice.

The rest of the dissertation is organized as follows. Chapter 2 provides a survey of related work, and Chapter 3 proposes the framework and motivation. Chapter 4 tackles the budget-constrained MapReduce workflow optimization problem in a cloud environment. Chapters 5 and 6 propose solutions to energy-efficient static mapping and dynamic scheduling of MapReduce workflows in a shared cluster, respectively. Chapter 7 concludes this dissertation and discuss future work. or local cluster computing

CHAPTER 2

RELATED WORK

This chapter is organized as follows. Section 2.1 provides a survey of related work on performance optimization of workflow mapping. Section 2.2 provides a survey of related work on energy-efficient workflow scheduling and job scheduling in Hadoop.

2.1 Performance Optimization

2.1.1 Workflow Scheduling Algorithms

We conduct a survey of related work on workflow performance optimization.

Classic Workflow Scheduling Algorithms Task scheduling or job mapping for workflows has been investigated extensively in the literature in the past decade [32, 33, 58, 15, 74]. Many heuristics have been proposed to minimize the workflow makespan (i.e., execution time) in grids, such as heterogeneous earliest finish time (HEFT) [68], and hybrid balanced minimum completion time (HBMCT) [62]. HBMCT first assigns weights to the nodes and edges of a workflow graph, and then partitions the nodes into ordered groups and schedules independent tasks within each group. These scheduling algorithms have been demonstrated to be very effective in makespan minimization in their targeted simulation or experimental settings.

Traditional Workflow Scheduling Algorithms in Clouds In [9], Abrishami *et al.* designed a QoS-based workflow scheduling algorithm based on partial critical paths (PCP) in SaaS clouds to minimize the cost of workflow execution within a user-defined deadline. As many existing critical-path heuristics, they schedule jobs on the critical path first to minimize the cost without exceeding their deadline. PCP are then formed ending at those scheduled jobs, and each PCP takes the start time of the scheduled

critical job as its deadline. This scheduling process continues recursively until all jobs are scheduled. In [53, 52], Mao *et al.* investigated the automatic scaling of clouds with budget and deadline constraints and proposed scaling-consolidation-scheduling (SCS) with virtual machines (VMs) as basic computing elements. In [34], Hacker proposed a combination of four scheduling policies based on an on-line estimation of physical resource usage. For the general area of workflow optimization in clouds, there are efforts towards several aspects. An important optimization issue on the user side is to achieve accurate estimation for cloud resources that would be needed, and make cost-effective provisioning. The work in [36] provide an automatic way to generate resource specification to help user make optimal cloud resource provisioning request. To store the large amount of data, the data placement problem is investigated through clustering [78]. The initial clustering sets the placement of initial data that the workflow requires and the intermediate data generated during the execution are clustered on line according to previous data clusters. The performance of distributed workflows execution in heterogenous network environments has been investigated in [74] with an analytical cost model aimed at minimizing end-to-end delay through optimal mapping scheme.

Budget-constrained Workflow in Grids There exist a relatively limited number of workflow scheduling efforts with a budget constraint in utility grids such as [76, 63, 62, 58]. In [63], Sakellariou *et al.* proposed two approaches, namely, LOSS and GAIN, to schedule DAG-structured workflow applications in grid environments. They start from a standard DAG schedule (such as HEFT or HBMCT) and a least-cost schedule, respectively, and reschedule the workflow to meet the budget constraint. In [77], Yu *et al.* performed a cost-based scheduling process through workflow task partitioning and deadline assignment to meet a user-defined deadline with the minimal cost. The workflow tasks are first categorized into synchronization tasks and simple

tasks according to the number of their parent and child tasks. Interdependent simple tasks that are executed sequentially are then grouped into branches connected by synchronization tasks. The overall deadline specified by the user is divided into sub-deadlines over the task partitions in proportion to their minimal processing time. However, the above work does not consider the virtualization characteristics of cloud environments.

Budget-constrained Workflow in Clouds Several recent efforts address workflow scheduling that takes both cost and delay into consideration in cloud environments [81, 13, 40]. BHEFT [81] by Zheng *et al.* extended HEFT to include budget constraint, and Arabnejad *et al.* proposed HBCS [13] to minimize workflow makespan under a cost constraint. Both BHEFT and HBCS adjust their VM/processor selection parameter based on the current budget usage at each step. Rodriguez *et al.* proposed a combined resource provisioning and scheduling strategy for executing scientific workflows in IaaS clouds to minimize the overall execution cost while meeting a user-defined deadline [60]. They designed a particle swam optimization-based approach that incorporates basic IaaS cloud principles such as a pay-as-you-go model, without considering the data transfer cost between data centers. Wu *et al.* formulated a job scheduling problem to minimize the end-to-end delay of a workflow under a user-specified financial constraint in a single cloud and designed a heuristic solution [73]. Also, Wu *et al.* further consider the performance optimization of a big-data workflow in multi-cloud environments under a budget constraint [72]. Jiang *et al.* addressed two main challenges in executing large-scale workflow ensembles in public clouds, i.e., execution coordination and resource provisioning. They developed DEWE v2, a pulling-based workflow management system with a profiling-based resource provisioning strategy, and demonstrated the effectiveness of their provisioning method in terms of both cost and deadline compliance [40]. The

above work is focused on performance optimization in clouds, without considering big data workflows in Hadoop.

Budget-constrained Workflows in Hadoop Systems Research efforts on scheduling a batch of MapReduce jobs under a budget constraint in Hadoop systems are still quite limited. Sandholm *et al.* proposed the dynamic priority parallel task scheduling algorithm, referred to as DPSS, for heterogeneous Hadoop clusters [65]. Our work differs from DPSS in two aspects: i) We consider sufficient capacities in clouds with different VM types to minimize the makespan within a given budget, while DPSS allows dynamic capacity distribution across concurrent users based on user preferences. ii) We aim to optimize the mapping of a workflow with MapReduce jobs from a global perspective, while DPSS attempts to optimize the budget on a per-job basis by allowing users to adjust their spending over time. Wang *et al.* modeled a batch of MapReduce jobs as a multi-stage fork-and-join workflow with no precedence in each stage and proposed one pseudo-polynomial optimal solution and two heuristics for task-level scheduling to minimize the maximum completion time under a budget constraint on heterogeneous VMs [70, 71]. In their mapping model, each task in a job is mapped onto a different VM instance; while in our mapping model, each job with adaptive task partitioning is mapped onto a set of shared VM instances with identical processing speed. In their algorithms, the number of stages has a significant impact on the optimality and time complexity; while in our algorithm, we do not divide a workflow into stages for performance improvement. Huang *et al.* designed a system, Cumulon, to help users rapidly develop and strategically deploy matrix-based big-data analysis programs in clouds according to time/budget constraints [35]. Different from Cumulon, our work is focused on the design of big-data workflow mapping algorithms in Hadoop/YARN.

2.1.2 Workflow Mapping with Malleable Jobs

We conduct a survey of related work on workflow mapping with malleable jobs. Several efforts have been made to minimize the maximum or total completion time of malleable jobs with execution precedences [56, 51, 39, 18]. Jansen *et al.* proposed a two-phase approximation algorithm with an approximation ratio of 3.291919 for scheduling workflows with malleable jobs [38, 39]. Chen *et al.* considered the scheduling of malleable jobs under general precedence constraints to find a minimum makespan, assuming that i) the processing time and the workload of a malleable job are non-increasing and non-decreasing, respectively, as the number of assigned processors increases, and ii) the workload function is convex with respect to the processing time. They proposed a polynomial-time approximation algorithm with an approximation ratio of 3.4142, which leads to an approximation ratio of 2.9549 when the processing time is strictly decreasing as the number of assigned processors increases [18]. The above work is focused on the theoretical analysis of malleable computing models. Our work attempts to minimize the makespan of a workflow with moldable jobs.

2.2 Energy Efficiency

2.2.1 Energy Efficiency in Hadoop Systems

Energy-efficient Data Placement A large number of research efforts have been made to optimize the data replication scheme in Hadoop distributed file system (HDFS) so that data nodes can be turned off without affecting data availability. To allow scale-down of an operational Hadoop cluster, Leverich *et al.* introduced the notion of a covering subset (CovSet) for HDFS, a small subset of machines, within which one replica of every block is stored [11]. This subset remains fully powered to preserve data availability while the rest can be turned off during low utilization periods. Lang *et al.* proposed the all-in strategy (AIS) that turns off all servers for

energy saving and turns on all servers to accommodate all tasks as fast as possible when the task queue is large enough. They demonstrated the superiority of AIS compared to CovSet in terms of response time and energy cost when the transition time of nodes to and from a low power state is relatively small compared with the total workload execution time [44]. Amur *et al.* proposed to maintain the primary replica of each data block on the primary nodes that are always active and store B/n secondary replicas on the n-th node on the expansion-chain (B is the total number of replicas), which denotes the order in which nodes must be turned on/off to scale performance up/down to support the equal-work layout for power-proportionality [12]. Chen *et al.* developed BEEMR, an energy-efficient MapReduce workload manager motivated by an empirical analysis of real-life traces of MapReduce workloads from Facebook [20]. The key insight is that interactive jobs often operate on a small fraction of data, and thus can be served by a small pool of dedicated machines, while jobs that are less time sensitive can run in a batch manner on the rest of the cluster. Energy savings come from aggregating the execution of less time-sensitive jobs in the batch zone to achieve high utilization, and transitioning idle machines in the batch zone to a low-power state. These techniques showed dramatic improvements in energy saving at the file system level. Our research on job scheduling is orthogonal to these efforts, and hence adds an additional level of energy efficiency to Hadoop systems.

Energy-efficient MapReduce Job Scheduling *Dynamic Voltage Frequency Scaling (DVFS)*: The DVFS technology has been widely adopted for energy saving in computing systems. Bampis *et al.* focused on the minimization of the total weighted completion time for a set of MapReduce jobs under a given energy constraint, and used a linear programming relaxation method to derive a polynomial-time constant-factor approximation algorithm [16]. Li *et al.* proposed an SLA-aware energy-efficient scheduling scheme that dynamically changes the CPU frequency for upcoming tasks

given the slack time between the actual execution time of completed tasks and the expected completion time of the MapReduce application in YARN [50].

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 VM placement algorithms to minimize the cumulative machine uptime of all 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 [17]. They also proposed techniques that dynamically scale MapReduce clusters to further improve energy consumption while ensuring that jobs meet or improve their expected runtimes.

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) [55]. Sharma *et al.* designed a dynamic scheduler for interactive and batch MapReduce jobs in hybrid physical and virtual environments to boost resource utilization and energy saving through workload consolidation based on virtualization and avoid virtualization-incurred overhead by executing "heavy" jobs immediately on PMs [67]. 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 [22]. The use of the ant colony algorithm in the Hadoop scheduler is based on an assumption that there exist a large number of homogeneous tasks in a MapReduce job. However, an excessively large number of tasks in a parallel job may incur very high overhead (compared with

the payload itself), hence leading to a significant waste of energy and delaying the job completion time.

Renewable Energy: Several efforts were focused on utilizing renewable energy in the operation of Hadoop clusters. Goiri *et al.* proposed a framework, GreenHadoop, for a data center powered by renewable (green) energy from a photovoltaic solar array 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 [30]. GreenHadoop dynamically schedules MapReduce jobs to minimize brown energy consumption by postponing background computations within their time bounds to run on the green energy supply that is not always available. 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 [23]. They formulated the job scheduling problem as an online optimization problem of minimizing the penalty of job deadline misses and solved it using a receding horizon control algorithm. To aid the control, they designed a self-learning model to estimate job completion times and used a simple but effective model to predict future resource availability. Despite the salient features for energy cost saving enabled by mixed energy supplies, at present there is no mature technology to support seamless switch between green and brown energy supplies or bring down the cost for storing renewable energy in such MapReduce frameworks.

Overhead Reduction: A few efforts were devoted to workload overhead reduction for energy saving. Sharma *et al.* designed a dynamic scheduler for interactive and batch MapReduce jobs in hybrid physical and virtual environments to boost resource utilization and energy saving through workload consolidation based on virtualization and avoid virtualization-incurred overhead by executing "heavy" jobs immediately on

PMs [67]. 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 [48]. Their approach dynamically selects appropriate preemption mechanisms based on the progress of a task and its suspend-resume overhead to improve job response time and reduce energy consumption. As opposed to preemptive scheduling of interactive applications in their work, our work is focused on energy saving in non-preemptive scheduling of MapReduce jobs in the background.

2.2.2 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 [57] by migrating active VMs onto energy-efficient PMs in time [75] or consolidating applications with complementary resource requirements [82]. 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 [82].

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 [47, 45], and designed two static energy-conscious workflow scheduling algorithms based on DVFS in heterogeneous distributed systems [46]. In [49], 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 [80], and designed a Pareto-based bi-objective genetic algorithm to achieve low energy consumption and high system reliability for static workflow scheduling [79]. 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 [83]. The above work only considers serial or rigid jobs in workflows, while our work is focused on moldable jobs in big data computing systems. Durillo *et al.* proposed a Pareto-based multi-objective (MOHET) static workflow scheduling algorithm as an extension to the heuristic, HEFT, capable of computing a set of tradeoff optimal solutions in terms of makespan and energy efficiency [26, 27]. They used an empirical model based on knowledge extracted from historical executions of real workflow tasks to compare the energy consumption and execution time of workflows of different shapes and sizes in heterogeneous parallel systems with different static and dynamic energy consumption. Krish *et al.* proposed the first energy-efficient workflow scheduler in Hadoop, so called ϵSched, which profiles the performance and energy characteristics of applications on each hardware sub-cluster in a heterogeneous cluster to improve the application-resource match while ensuring energy efficiency and performance-related SLA goals [42]. Mao *et al.* designed GreenPipe, a scalable computational workflow system that runs MapReduce jobs on virtual Hadoop clusters, and proposed a power-aware scheduling algorithm in the workflow engine to optimize workflow execution in terms of execution time and energy consumption [54].

2.2.3 Malleable Job Scheduling

Some efforts have been made to minimize the completion time of a workflow comprised of malleable jobs [39, 56, 18, 51], but there exist relatively limited efforts on moldable/malleable job scheduling for energy efficiency. Sanders *et al.* designed a polynomial-time optimal solution and a fully polynomial-time approximation scheme (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 [64]. Different from these theoretical models, our work employs measurement-based power consumption models and performs workflow mapping to reduce the computing overhead and thus improve the energy efficiency of big data workflows. To the best of our knowledge, our work is among the first to study energy-efficient mapping of big data workflows comprised of moldable jobs in Hadoop systems.

CHAPTER 3

PERFORMANCE MODELING AND OPTIMIZATION FRAMEWORK

This chapter is organized as follows. Section 3.1 presents a layered workflow optimization architecture, within which big-data computing workflows are mapped and scheduled. Section 3.2 conducts an in-depth investigation into the performance computing model of moldable parallel jobs, exemplified by MapReduce programs, in terms of execution time and dynamic energy consumption.

3.1 Workflow Optimization Architecture

As illustrated in Figure 3.1, we consider a layered workflow optimization architecture in a data center deployed in two different environments. 1) In a single cloud environment, we consider three layers for big data workflow mapping: i) the big data workflow layer comprised of interdependent MapReduce jobs, each of which contains one or more map tasks and zero or more reduce tasks, ii) the VM layer representing a set of fully connected virtual machines, and iii) the server layer constituted by a number of physical machines connected via a local network [73]. Note that end users only knows the region of a VM, as the infrastructure in the bottom layer is invisible to them. We consider a many-to-one mapping scheme such that multiple tasks of the same job can be assigned to a shared VM instance. 2) In a shared PC cluster environment, we only consider two layers for big data workflow mapping/scheduling, i.e., i) the big data workflow layer and ii) the server layer, and consider a many-to-one mapping scheme such that multiple tasks from the same job or different jobs can be assigned to a shared PM. The interactions between these layers produce an integrated and intelligent workflow solution to model and optimize big-data scientific applications in resource sharing environments.

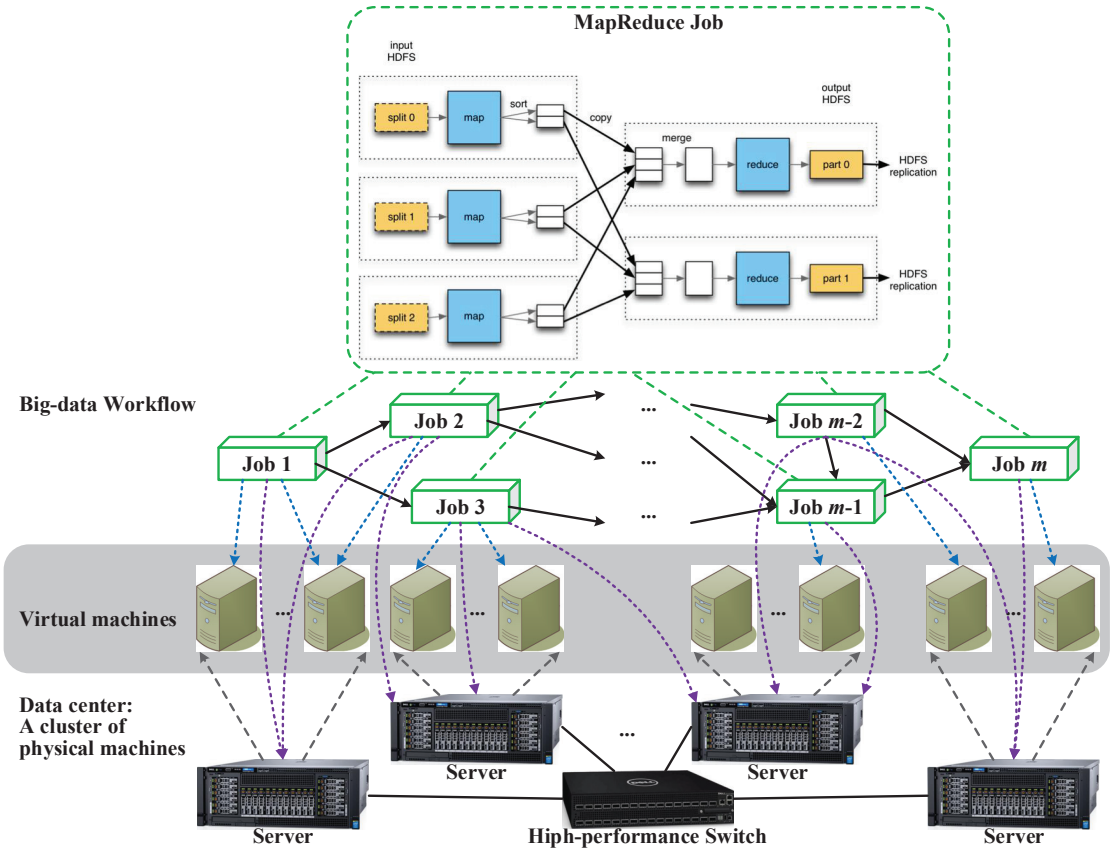


Figure 3.1 The architecture for MapReduce workflow optimization in a data center.

The top layer defines abstract scientific workflows comprised of data-intensive computing jobs with application-level functional and I/O descriptions. This layer also provides a unified web interface for users to compose, dispatch, and monitor domain-specific workflows while the rest of the system is made completely transparent to them. The simplest workflow may include only a chain of two jobs while a complex one may involve as many as thousands of jobs with intricate execution dependencies.

The bottom layer defines underlying physical system resources including large data repositories storing (simulated, observational, or experimental) high-resolution multimodal scientific datasets, high-speed network infrastructures provisioning high bandwidth for fast data transfer, and HPC facilities generating countless CPU cycles for expeditious data processing.

In a cloud environment, the top and bottom layers meet at the middle layer that defines a virtual overlay cluster through the following two operations:

- Resource virtualization: Build a virtual overlay cluster from the underlying computing resources. Each overlay node with estimated processing power corresponds to a VM in clouds.
- Workflow mapping and scheduling: Determine a workflow mapping scheme that assigns each task in each job in the workflow to an overlay node, and decide a task scheduling policy on each mapped node to optimize end-to-end workflow performance such as latency.

3.2 Performance Model of Moldable Jobs

In big data systems such as Hadoop/Spark, there arises a new challenge to optimize the mapping/scheduling of moldable parallel jobs. 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 [31]. This model is based on the well-known Brent's lemma, which states that the parallel execution of a job may achieve some speedup if the job is sufficiently large, but does not lead to superlinear speedups. The validity of this model has been verified by many real-life parallel programs on disparate high-performance computing platforms and will serve as a base of our mapping/scheduling solution for performance optimization and energy saving of big data workflows.

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 dynamic energy consumption in a PC cluster and job's financial cost in a cloud) increases. Hence, if the number of tasks does not exceed the maximum number of parallel tasks (executed simultaneously) supported by the system, the job execution time is the same as the

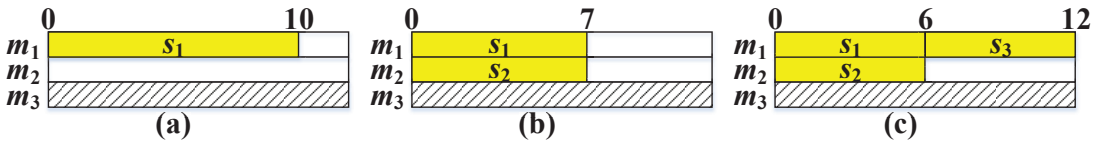


Figure 3.2 A moldable job.

task execution time and thus decreases as the number of tasks increases; otherwise, both the job execution time and energy consumption may increase with the number of tasks.

We consider a computing performance model where the total dynamic energy consumption (DEC), decided by the total workload, of a moldable parallel job increases and the execution time of each task decreases as the number of parallel tasks in the job increases. We present a numerical example in Figure 3.2 to illustrate the possibility of reducing the execution time and DEC of a moldable job by properly adjusting the number of parallel tasks. 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 one, two, or three 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 how the degree of parallelism affects job workload (or DEC) and execution time in big data computing applications, which lays down the foundation of this research.

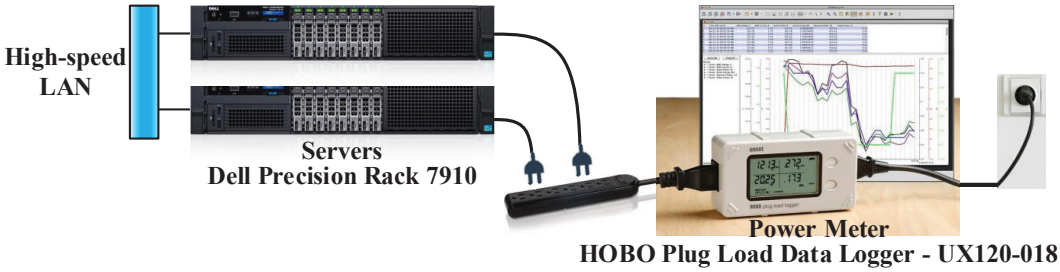


Figure 3.3 The experimental testbed for measuring energy consumption.

3.2.1 Experimental Settings

We set up a small-scale homogeneous cluster comprised of two Dell servers, each of which is equipped with two processors of Intel(R) Xeon(R) CPU E5-2630 v3 with 15MB cache and six cores of 2.4GHz, 16GB 2133MHz DDR4 RDIMM ECC memory, and 256GB 2.5inch serial ATA solid state drive. We install a power meter of 0.5% relative measurement errors with a measurement resolution of 1 watt, HOBO Plug Load Data Logger – UX120-018, to collect the active power/energy consumption of the entire cluster in the testbed, as shown in Figure 3.3. 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 [2]. 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 [8] and use the example MapReduce programs in Apache Hadoop 2.7.3 as 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). Initially, each dataset is stored in multiple separate files. 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. In fact, the number of reduce keys would affect the parallelism degree of reduce tasks. The first MapReduce job (i.e., PAS) has 29 reduce keys (i.e., airport names); the second MapReduce job (i.e., ATA) has 340 reduce keys (i.e., flight numbers); and the third MapReduce job (i.e., FCR) has 5 reduce keys (i.e., cancellation codes).

We repeatedly run each MapReduce program with and/or without the reducing phase for 10 times on our homogenous mini-cluster and measure the corresponding DEC and execution time of the mapping and reducing phases in each MapReduce job. To adjust the number of mappers and reducers in each MapReduce job, we set the properties of "mapreduce.input.fileinputformat.split.minsize", "mapreduce.input.fileinputformat.split.maxsize", and "mapreduce.job.reduces" to be different values in the configuration file. To make the map tasks homogeneous, we divide the entire input data evenly by properly adjusting the split size.

3.2.2 Experiment-based Model Validation

We tabulate the average DEC and execution time of the mapping phase of each statistical MapReduce application with a varying number of map tasks in Table 3.1. We further plot the average DEC and execution time of each MapReduce benchmark and the mapping phase of each statistical application with different numbers of map tasks in Figures 3.4 and 3.5, respectively, for a visual comparison. 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 [41] increases up to 23, which is the largest number of map tasks supported simultaneously by the system. When the number of map tasks exceeds 23, the job execution time

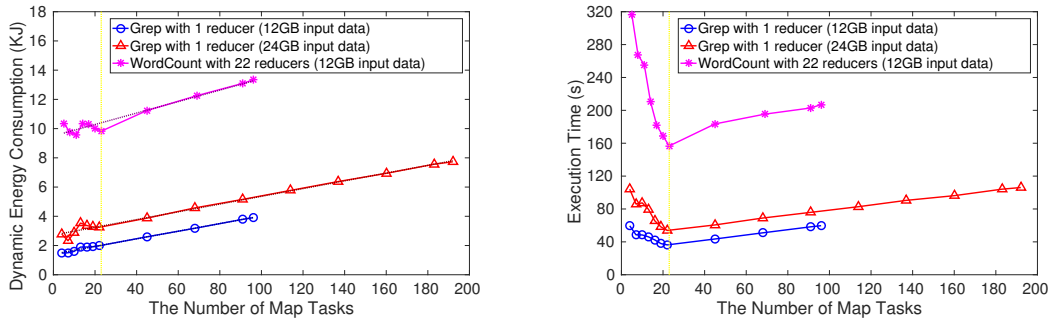
Table 3.1 The Execution Time and DEC of Mapping vs. the Number of Splits

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

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

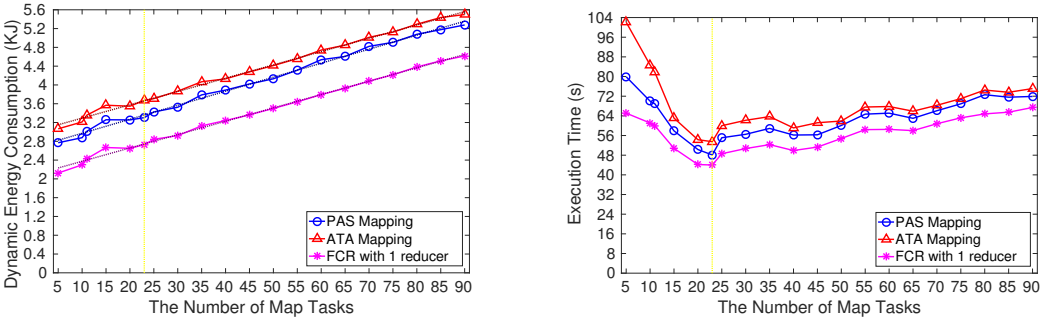
exhibits a fluctuant increase as the number of waves in the mapping phase increases from 2 to $\lceil 96/23 \rceil = 5$ for Grep12 and WordCount12, $\lceil 192/23 \rceil = 9$ for Grep24, or $\lceil 90/23 \rceil = 4$ for our statistical applications. The fluctuation in Figure 3.5(b) is due to the fact 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., $f_1(k) = 0.0181k + 0.9819$, $f_2(k) = 0.0097k + 0.9903$, and $f_3(k) = 0.0041k + 0.9959$, $f_4(k) = 0.0108k + 0.965$, $f_5(k) = 0.0092k + 0.982$, and $f_6(k) = 0.0134k +$



(a) (b)

Figure 3.4 Benchmarks: (a) the DEC vs. the number of map tasks; (b) the execution time vs. the number of map tasks.



(a) (b)

Figure 3.5 The mapping phase of our statistical application: (a) the DEC vs. the number of map tasks; (b) the execution time vs. the number of map tasks.

0.9835, respectively, which serves as the base of parameter setting in our simulation in Subsection 6.5.2. The system log shows that there are 0 to 3 killed/resumed map tasks in each job execution, which explains the variations in Figures 3.4(a) and 3.5(a).

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 3.2. Such a trend in reducing does not seem to be as obvious as that in mapping for two main reasons: i) There exists critical reduce-skew [43] for a small number of reduce keys. ii) Since the workload of reducing is much less than that of mapping in our statistical MapReduce jobs, the measurement errors for reduce tasks in Table 3.2 are relatively

Table 3.2 The Execution Time and DEC of Reducing vs. the Number of Reducers

The Number of Reducers	Reducing in Job PAS		Reducing in Job 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

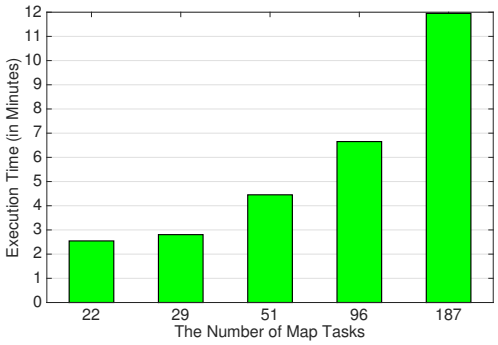


Figure 3.6 The execution time of a MapReduce job vs. the number of tasks.

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.

To further validate the computing performance model in different scenarios, we run the third MapReduce program (i.e., FCR) with 22 separate input files in HDFS on another older computer server equipped with two processors of Intel(R) Xeon(R) CPU E5-2630 with six cores of 2.3GHz and 64GB memory. The program execution time is measured and plotted in Figure 3.6, which shows that the execution time of this MapReduce job increases as the number of tasks increases when the server is

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

CHAPTER 4

PERFORMANCE OPTIMIZATION OF MAPREDUCE WORKFLOW MAPPING IN CLOUDS

This chapter is organized as follows. Section 4.1 introduces the background and significance of performance optimization of MapReduce workflow mapping in clouds. Section 4.2 formulates a big data workflow mapping problem. We design an FPTAS for a special case with a pipeline-structured workflow in Section 4.3, and a heuristic for a generalized problem in Section 4.4. Section 4.5 presents the performance evaluation. Section 4.6 concludes our work.

4.1 Introduction

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

In recent years, an increasing number of big data workflows have migrated to clouds, which has reaped the benefit of resource virtualization but meanwhile has also brought many new challenges for workflow execution and performance optimization [73]. Cloud computing makes computing a utility such that one pays for only those cloud resources that are truly needed and used [52]. Hence, to support cost-effective execution of big data workflows in clouds, researchers are now facing the challenge of reducing financial cost in addition to meeting traditional performance optimization goals [73].

In this chapter, we design a big-data workflow mapping model, where a job scheduler adaptively partitions each MapReduce job into a certain number

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

4.2 Problem Formulation

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

4.2.1 Cost Models

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

the price is a linear function with respect to the capacity of the VM types in a single class, and the capacity of a VM type is α_j ($\alpha_j > 1, \alpha_j \in \mathbb{Z}$) times that of the next lower VM type, as in Amazon EC2 pricing model.

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

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

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

Time Model Since contiguous tasks can time-share a common VM instance over different periods, the total VM initialization time for large-scale workflows could be negligible. Since our work targets big data applications in cloud environments in a single data center where PMs are interconnected via high-speed links, we do not specifically consider data transfer time. Hence, the time cost of a task $r_{k,l}$ running on a VM instance of a type in class C_i can be simplified as the execution time of $r_{k,l}$ on a VM in C_i, i.e., $t_{i,k,l} = w_{k,l}(L_k)/(u_{i,k} \cdot s_i)$, and so is the time cost $t_{i,k}(L_k)$ of job v_k if all L_k tasks start to run on VM instances in class C_i at the same time. To avoid financial waste, at least one task is executed as soon as a VM instance is activated, and the instance should be stopped if no suitable tasks can run on it immediately.

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

Data transferred between Amazon EC2 instances located in different Availability Zones in the same Region will be charged Regional Data Transfer. Data transferred between AWS services in different regions will be charged as Internet Data Transfer on both sides of the transfer. Usage for other Amazon Web Services is billed separately from Amazon EC2. Availability Zone Data Transfer: $0.00 per GB all data transferred between instances in the same Availability Zone using private IP addresses. Regional Data Transfer: $0.01 per GB all data transferred between instances in different Availability Zones in the same region.

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

4.2.2 Problem Definition

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

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

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

Table 4.1 Notations Used in the Cost Models

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

subject to the budget and precedence constraints:

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

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

4.2.3 Computational Complexity Analysis

We first consider a special case of BCWM4C as follows: given a VM type of price p with sufficient memory and 5 CPU cores of speed s, and K independent serial jobs $\{v_k\}$ with CPU-burst workload w_k, does there exist a feasible non-preemptive scheduling scheme under a budget constraint B such that the makespan is no more than $T = B/p$? This special case has been proved to be strongly NP-hard in [25], so is the general BCWM4C problem, which, with a polynomially bounded objective function, has no FPTAS unless $P = NP$ [69].

4.3 A Pipeline-structured Workflow on VMs in a Single Class

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

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

4.3.1 Computational Complexity Analysis

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

Definition 3. ***Two-Choice Knapsack****: Given K classes $S_1, S_2, \ldots, S_K$ of items to pack in a knapsack of capacity Q, where each class has exactly two items $d_{k,l} \in S_k$ ($l \in \{1, 2\}, k = 1, 2, \ldots, K$), each of which has a value $o_{k,l}$ and a weight $q_{k,l}$, is there*

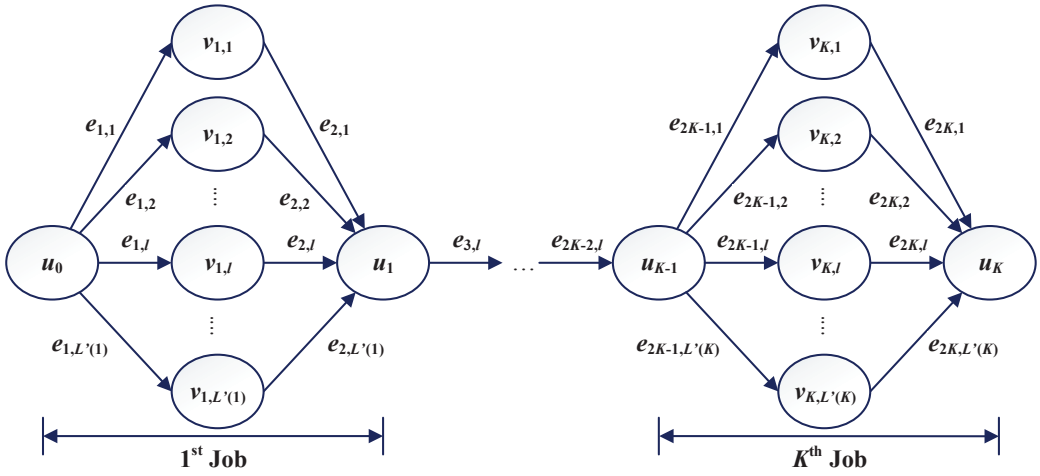


Figure 4.1 A constructed network corresponding to a workflow with a pipeline structure, where $L'(1) = L'_1$ and $L'(K) = L'_K$.

a choice of exactly one item from each class such that the total value is no less than O and the total weight does not exceed capacity Q?

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

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

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

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

We construct an instance of BCWM4C-PWSC as follows. Let $n_1 = 1$, $n_2 = 2$, $m_2 = 2m_1$, $p_1 = (o_{k,1} - o_{k,2})/(2q_{k,2} - q_{k,1})$, $p_2 = 2p_1$, $v_k = S_k$, $L'_k = 2$, $w_k(1) = q_{k,1}u_k s$, $w_k(2) = 2q_{k,2}u_k s$, $T = Q$, and $B = \sum_{1 \leq k \leq K} O_k - O$, where $O_k = (2q_{k,2}o_{k,1} -$

$q_{k,1}o_{k,2})/(2q_{k,2} - q_{k,1})$. It is obvious that this construction process can be done in polynomial time.

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

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

4.3.2 Approximation Algorithm

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

Definition 4. ***Restricted Shortest Path****: Given a directed graph $\mathbb{G}(\mathbb{V}, \mathbb{E})$, where each edge $e \in \mathbb{E}$ is associated with cost $c(e) > 0$ and length $d(e) > 0$, vertices $s, t \in \mathbb{V}$, and cost constraint $C > 0$, we wish to find the shortest simple path from s to t in $\mathbb{G}$ with the total cost not exceeding C.*

Given an instance of BCWM4C-PWSC, we construct an instance of RSP according to the pipeline as follows. As illustrated in Figure 5.1, the network graph G consists of $\mathbb{V} = \{v_{k,l} | k = 1, \ldots, K, l = 1, \ldots, L'_k\} \cup \{u_0, u_k | k = 1, \ldots, K\}$ with a source u_0 and a destination u_k, and $\mathbb{E} = \{e_{2k-1,l}, e_{2k,l} | k = 1, \ldots, K, l = 1, \ldots, L'_k\}$, where $e_{2k-1,l} = (u_{k-1}, v_{k,l})$ and $e_{2k,l} = (v_{k,l}, u_k)$. Then, we calculate the execution time of job v_k with l tasks as $t_k(l) = w_k(l)/(l \cdot s \cdot u_k)$, and accordingly its expense as $e_k(l) = t_k(l)p_k(l)$, where $p_k(l)$ is the least expensive price for executing l tasks in v_k in parallel.

Algorithm 1: PMMM()

1: Construct a DAG $\mathbb{G}(\mathbb{V}, \mathbb{E})$ for a given pipeline-structured workflow as shown in Figure 5.1, and assign cost $e_k(l)$ and length $t_k(l)$ to edge $e_{2k-1,l}$ and zero cost and zero length to edge $e_{2k,l}$;

2: Use FPTAS in [28] to find the shortest path from u_0 to u_K under cost constraint B with approximate rate ϵ_1 and convert it to mapping scheme

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

To minimize makespan under a budget constraint In Algorithm 1, we assign the cost $c(e)$ and length $h(e)$ of each edge $e \in \mathbb{E}$ as $c(e_{2k-1,l}) = e_k(l)$, $l(e_{2k-1,l}) = t_k(l)$, and $c(e_{2k,l}) = l(e_{2k,l}) = 0$, and set the cost constraint on a path from u_0 to u_K to be budget B. As a result, the shortest length in RSP is exactly the minimum makespan in BCWM4C-PWSC, and if $v_{k,l}$ is on the solution path to RSP, the k-th job has l tasks. If the FPTAS in [28] is used to solve RSP, BCWM4C-PWSC finds a feasible solution within the shortest makespan multiplied by $(1 + \epsilon_1)$ in time $O(K^2 L'^2/\epsilon_1)$, where $L' = \max_{1 \le k \le K} L'_k$. If the FPTAS in [28] is used to solve RSP in the topology in Figure 5.1, BCWM4C-PWSC finds a feasible solution within the shortest makespan multiplied by $(1 + \epsilon_1)$ in time $O(KL'(\log L' + 1/\epsilon_1))$, thanks to the special topology in Figure 5.1, where $L' = \max_{1 \le k \le K} L'_k$.

To minimize expense under a makespan constraint In Algorithm 10, we swap the cost and length of each edge in Algorithm 1, and set the path cost constraint to be makespan bound T. Similarly, the shortest length in RSP is exactly the minimum expense in BCWM4C-PWSC. If the FPTAS in [19] is used to solve RSP, BCWM4C-PWSC finds a solution with the least expense under the makespan

Algorithm 2: PMEM()

1: Construct a DAG $\mathbb{G}(\mathbb{V}, \mathbb{E})$ for a given pipeline-structured workflow as shown in Figure 5.1, and assign cost $t_k(l)$ and length $e_k(l)$ to edge $e_{2k-1,l}$ and zero cost and zero length to edge $e_{2k,l}$;

2: Use FPTAS in [19] to find the shortest path from u_0 to u_K under cost constraint T with approximate rate ϵ_2 and convert it to mapping scheme

constraint multiplied by $(1+\epsilon_2)$ in time $O((K^2L')/\epsilon_2)$, thanks to the special topology in Figure 5.1.

4.4 Algorithm for Arbitrary Workflows on VMs in Different Classes

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

4.4.1 An Overview of BAWM

For the sake of clarification, we list some variables and functions used in BAWM in Table 4.2. Each job v_k is associated with a set of pairs of the number $L_{k,n}$ of tasks ($L_{k,n} \in [1, L'_k]$) and the class $C_{k,n}$ of assigned VM types ($C_{k,n} \in \{C_i\}$). Each pair corresponds to certain execution time $t_{k,n} = t_k(L_{k,n}, C_{k,n})$ and expense $e_{k,n} = e_k(L_{k,n}, C_{k,n})$ as listed in Table 4.3. The quadruples $\{(t_{k,n}, e_{k,n}, L_{k,n}, C_{k,n})\}$ in Table 4.3 are sorted in the ascending order of execution time, and are referred to as the time-expense table (TET) tet_k of job v_k. Here, each quadruple corresponds to an execution option of job v_k, so if its execution time and expense are both larger than those of another one, it would be deleted from tet_k;

In Algorithm 3, BAWM first builds a time-expense table for each job by calling $bldTET()$, and then considers two special cases: i) If the input workflow is a pipeline,

Table 4.2 Variables and Functions Used in Algorithm Design

Notations	Definitions
$e_{\min}$	the smallest allowable budget for a feasible workflow mapping scheme
$e_{\max}$	the smallest sufficient budget for the fastest execution of all jobs
$tet \leftarrow bldTET$ $(V, \{C_i\})$	Build TETs tet for all jobs in V according to a set V of jobs and a set of classes $\{C_i\}$ of VM types
$(t, e, optIdx)$ $\leftarrow save$ $(f, tet, optIdx)$	Reclaim unnecessary budget allocation for workflow f without makespan increase and then return makespan t, expense e, and the option indices $optIdx$ of all jobs in the latest mapping
$(g, t) \leftarrow getCG$ $(f, tet, optIdx)$	Compute the critical subgraph g and makespan t of workflow f based on the TETs tet and option indices $optIdx$ of all jobs
$t \leftarrow calMS$ $(f, tet, optIdx)$	Calculate the makespan of workflow f based on the TETs tet and option indices $optIdx$ of all jobs
$e \leftarrow calExp$ $(f, tet, optIdx)$	Calculate the expense e of workflow f based on the TETs tet and option indices $optIdx$ of all jobs

Table 4.3 Time-Expense Table tet_k of Job v_k

$t_{k,1}$	$<$	$t_{k,2}$	$<$	$\ldots$	$<$	$t_{k,n}$	$<$	$\ldots$	$<$	$t_{k,N}$
$e_{k,1}$	$>$	$e_{k,2}$	$>$	$\ldots$	$>$	$e_{k,n}$	$>$	$\ldots$	$>$	$e_{k,N}$
$L_{k,1}$		$L_{k,2}$		$\ldots$		$L_{k,n}$		$\ldots$		$L_{k,N}$
$C_{k,1}$		$C_{k,2}$		$\ldots$		$C_{k,n}$		$\ldots$		$C_{k,N}$

Algorithm 1 is used to select an execution option for each job (Lines 5-7). Due to different execution speeds of a task running on VMs in different classes, Algorithm 1 is merely a heuristic, not an approximation algorithm. ii) If the budget approaches the smallest sufficient budget $e_{\max}$ for the fastest execution of all jobs, BAWM keeps makespan the shortest and reduces the expense of jobs on non-critical paths by calling $save()$ to search for the minimal expense (Lines 8-12). In $save()$, we calculate the ratio of the expense decrease over the time increase for each slower execution option

Algorithm 3: BAWM()

Input: A workflow $f(G(V, A), B)$ and a set $Y = \bigcup_i C_i$ of VM types

Output: a makespan t and a remaining budget b

1: $tet \leftarrow bldTET(V, \{C_i\})$;

2: Find the minimum expense as $e_{\min}$ when each job selects the least expensive execution option and the maximum expense as $e_{\max}$ when each job selects the fastest execution option;

3: **if** $B < e_{\min}$ **then**

4: Exit with an error.

5: **if** f is a pipeline **then**

6: Use Algorithm 1 to compute the option index $optIdx[k]$ of each job v_k;

7: **return** $(calMS(f, tet, optIdx), B - calExp(f, tet, optIdx))$;

8: **for all** $v_k \in V$ **do**

9: $optIdx[k] \leftarrow$ the index of the first (fastest) option in tet_k;

10: $(t, e, optIdx) \leftarrow save(f, tet, optIdx)$;

11: **if** $e \leq B$ **then**

12: **return** $(t, B - e)$;

13: **if** $B \leq e_{\min} + \delta \cdot (e_{\max} - e_{\min})|_{\delta=0.07}$ **then**

14: **for all** $v_k \in V$ **do**

15: $optIdx[k] \leftarrow$ the index of the last (least expensive) option in tet_k;

16: **return** $CGG(f, B - e_{\min}, tet, optIdx)$;

17: $(b, optIdx) \leftarrow BSMCEM(f, tet)$;

18: **return** $CGG(f, b, tet, optIdx)$.

Algorithm 4: bldTET()

Input: all the jobs $\{v_k\}$ in a workflow f and VM type classes $\{C_i\}$

Output: time-expense tables $\{tet_k\}$ for all jobs $\{v_k\}$

1: **for all** $v_k \in V$ **do**

2: $tet_k \leftarrow \emptyset$;

3: **for all** $C_i \subset Y$ **do**

4: **for** $L \leftarrow 1$ to L'_k **do**

5: Calculate the execution time $t_k(L, C_i)$ and the minimum expense $e_k(L, C_i)$ for job v_k with L tasks running on a set of VM instances in class C_i in parallel;

6: Add $(t_k(L, C_i), e_k(L, C_i), L, C_i)$ into tet_k.

7: Sort quadruples in tet_k in the ascending order of $t_k(L, C_i)$;

8: For each quadruple, if its execution time and expense are both larger than those of another one, delete it from tet_k;

9: **return** $\{tet_k\}$.

of each non-critical job, and then downgrade the job with the highest ratio to the corresponding execution option one at a time.

As shown in Algorithm 3, the key idea of BAWM is as follows. Each workflow mapping consists of two components: critical-subgraph-greedy (CGG) and binary search based on makespan-constrained expense minimization (BSMCEM). A CP is the longest execution path in a workflow, which can be calculated in linear time. Here, we define the union of all critical paths as a critical subgraph, which can also be calculated in linear time. CGG is designed to find a local optimal solution; BSMCEM is designed to find a feasible solution near a global optimal solution with some remaining budget. Therefore, if the budget approaches the smallest allowable budget $e_{\min}$ for a feasible workflow mapping scheme, BAWM uses $CGG()$ in Algorithm 6

Algorithm 5: save()

Input: a workflow $f(G(V, A))$, time-expense tables $\{tet_k\}$ and the option index $optIdx[k]$ of job v_k

Output: makespan t, expense e, and the indices $optIdx$ of options for all jobs

1: **while** $b > 0$ **do**
2: $(g, t) \leftarrow getCG(f, tet, optIdx)$; $g' \leftarrow f - g$;
3: **if** $g' = \emptyset$ **then**
4: break;
5: **for all** $v_k \in g'$ **do**
6: $i \leftarrow optIdx[k]$;
7: **for** $l = i+1$ to $tet_k.size()$ **do**
8: $optIdx[k] \leftarrow l$; $t' \leftarrow calMS(f, tet, optIdx)$;
9: **if** $t' = t$ **then**
10: $etr(k, l) \leftarrow (tet_k[i].e\text{-}tet_k[l].e)/(tet_k[l].t\text{-}tet_k[i].t)$;
11: $optIdx[k] \leftarrow i$;
12: Find job k^* and option index l^* with the maximum $etr(k^*, l^*)$;
13: **if** $etr(k^*, l^*) > 0$ **then**
14: $optIdx[k^*] \leftarrow l^*$;
15: **else**
16: break;
17: $e \leftarrow calExp(f, tet, optIdx)$;
18: **return** $(t, e, optIdx)$.

to search for the shortest makespan (Lines 13-16). Otherwise, BAWM first uses $BSMCEM()$ in Algorithm 7 to find a near-optimal feasible solution, and then starts $CGG()$ with this solution and utilizes its remaining budget to achieve a better solution (Lines 17-18).

Algorithm 6: CGG()

Input: a workflow $f(G(V, A))$, remaining budget b, time-expense tables $\{tet_k\}$ and the option index $optIdx[k]$ of each job v_k

Output: makespan t and remaining budget b^*

1: $t \leftarrow calMS(f, tet, optIdx)$; $b^* \leftarrow b$;
2: **while** $b > 0$ **do**
3: $(g, t) \leftarrow getCG(f, tet, optIdx)$;
4: **if** Remaining budget b cannot be used to speed up any job in g **then**
5: break;
6: $S \leftarrow \{v \in g |$ all critical paths traverse $v\}$;
7: **for all** $v_k \in S$ **do**
8: $i \leftarrow optIdx[k]$;
9: **for** $n = i$-1 to 1 **do**
10: $optIdx[k] \leftarrow n$; $t' \leftarrow calMS(f, tet, optIdx)$;
11: $\Delta e(k, n) \leftarrow e_{k,n} - e_{k,i}$;
12: **if** $\Delta e(k, n) < b$ **then**
13: $ter(k, n) \leftarrow (t - t')/\Delta e(k, n)$;
14: $optIdx[k] \leftarrow i$;

15: Find job k_1 and option index n_1 with the maximum $ter(k_1, n_1)$;

16: **if** $ter(k_1, n_1) > 0$ **then**

17: $optIdx[k_1] \leftarrow n_1$; $b \leftarrow b - \Delta e(k_1, n_1)$; continue;

18: $b^* \leftarrow b$; $optIdx^* \leftarrow optIdx$;

19: **for all** $v_k \in g - S$ **do**

20: $n \leftarrow optIdx[k]$;

21: $ter'(k, n-1) \leftarrow (t_{k,n} - t_{k,n-1})/(e_{k,n-1} - e_{k,n})$;

22: Find job k_2 and option index n_2 with the maximum $ter'(k_2, n_2)$;

23: **if** $ter'(k_2, n_2) > 0$ **then**

24: $optIdx[k_2] \leftarrow n_2$; $b \leftarrow b - \Delta e(k_2, n_2)$;

25: **return** (t, b^*).

4.4.2 Critical-subgraph Greedy

In Algorithm 6, we allocate the remaining budget based on the currently selected option for each job to accelerate the workflow execution. We select those jobs all CPs traverse as critical points and upgrade their options to faster ones, with an attempt to maximize the impact of a local speed-up on the global end-to-end delay. For each such job, we first calculate its local maximum ratio of the makespan decrease over the expense increase and record the corresponding option index, and then select the job that achieves the global maximum ratio over the entire critical subgraph for rescheduling. If none of the jobs at the critical points can be accelerated, we upgrade one of the rest in the critical subgraph by one level with the same selection method as above.

4.4.3 BSMCEM

The CP plays a significant role in the performance optimization of workflow mapping. However, it is difficult to balance the expenses between the jobs on

the CP and the jobs on noncritical paths (NCPs) as the CP might change during the workflow mapping process. Towards this end, we first design an heuristic to solve another optimization problem, MAkespan-constrained Workflow Mapping for Expense Minimization (MAWMEM), which has the same decision version as

Algorithm 7: BSMCEM()

Input: A budget-constrained workflow $f(G(V, A), B)$ and TETs tet

Output: remaining budget b and the option indices $optIdx$ of all jobs

1: Find the minimum makespan as $t_{\min}$ when each job selects the fastest execution option and the maximum makespan as $t_{\max}$ when each job selects the least expensive execution option;

2: $t_F \leftarrow t_{\min}/(1+\epsilon_2)$; $t_S \leftarrow t_{\max}$;

3: $t' \leftarrow$ -1; // t' is the time constraint of the latest successful mapping

4: **while** $t_S - t_F > \Delta t$ **do**

5: $t \leftarrow (t_S + t_F)/2$; $(t^C, e, optIdx) \leftarrow MAWMEM(t, f, tet)$;

6: **if** $e > B$ **then**

7: **if** $t_F < t^C < t$ **then**

8: $t_F \leftarrow t^C$;

9: **else**

10: $t_F \leftarrow t$;

11: **else**

12: **if** $t_S > t^C$ **then**

13: $t_S \leftarrow t^C$; $t' \leftarrow t$;

14: **else**

15: break;

16: Cancel the mapping of all the jobs in V;

17: **if** $t' \geq 0$ **then**

18: $(t^C, e, optIdx) \leftarrow MAWMEM(t', f, tet)$;

19: $b \leftarrow B - save(f, tet, optIdx)$;

20: **else**

21: $b \leftarrow B - e_{\min}$;

22: **return** $(b, optIdx)$.

BCWM4C. The difference between BCWM4C and MAWMEM is that their objective and one of the constraints are swapped. The objective of MAWMEM is

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

and the first constraint of MAWMEM is

$$t^C \leq T.$$

Accordingly, the framework of BSMCEM in Algorithm 7 is as follows. Initially, we set the infeasible makespan t_F to be the minimum makespan $t_{\min}$ minus a tiny amount and the feasible makespan t_S to be the maximum makespan $t_{\max}$, and seek for workflow mapping with a minimal makespan by binary search (Lines 4-16). Then, BSMCEM searches for the maximal remaining budget with this minimal makespan fixed by calling $save()$.

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

Algorithm 8: MAWMEM()

Input: makespan constraint T, workflow $f(G(V, A))$, and TETs tet

Output: makespan t^C, expense e, the option indices $optIdx$ of all jobs

1: $t_k^{LF} \leftarrow +\infty$ for $\forall v_k \in f$; $t_K^{LF} \leftarrow T$ for the end job v_K in f;

2: $G' \leftarrow G$; // G' is a subgraph of all the unmapped jobs in G.

3: **while** $G' \neq \emptyset$ **do**

4: Find the critical path cp ending at a job with the earliest LFT in

5: G' according to $\{t_{i,k}(L'_k)|e_{i,k}(L'_k) = \min_{C_{i'} \subset Y} e_{i',k}(L'_k)\}$;

6: **if** $PM(cp, tet) =$ False **then**

7: $v_k \leftarrow$ the last mapped job in cp;

8: $D(v_k) \leftarrow \{$the downstream jobs of v_k in $G\}$;

9: **for all** $v_{k'} \in D(v_k)$ s.t. $t_{k'}^S < t_k^F$ **do**

10: Cancel the mapping of $v_{k'}$, and add it and its associated precedence constraints back to G';

11: $G' \leftarrow G' - \{v \in cp | v$ is mapped$\}$;

12: $t^C \leftarrow calMS(f, tet, optIdx)$; $e \leftarrow calExp(f, tet, optIdx)$;

13: **return** $(t^C, e, optIdx)$.

and temporary last finish time (TLFT) $t'_{LF}(v_k)$ can be calculated based on only its mapped preceding and succeeding jobs, respectively. The EST and LFT of a pipeline are the EST of its first job and LFT of its end job, respectively.

We consider MAWMEM and design a heuristic algorithm in Algorithm 8, whose key idea is as follows. Each workflow mapping consists of two components: iterative CP selection and pipeline mapping. The algorithm starts with computing an initial CP according to the execution time of each job with a maximum number of tasks

running in parallel in the least expensive VM class, followed by a pipeline mapping process. Then, it iteratively computes a CP with the earliest last finish time (LFT) from the remaining unmapped workflow branches based on the same execution time of a job as above and performs a pipeline mapping of the computed CP until there are no branches left. If the mapping of the last mapped job on the CP violates the precedence constraint with its downstream jobs, all these downstream jobs in violation would be unmapped. Note that the first job on each previous mapped CP

Algorithm 9: PM()

Input: a pipeline pl with its EST $pl.est$ and LFT $pl.lft$ and TETs tet

Output: a boolean variable to indicate whether mapped jobs in pl follow precedence constraints

1: Label the index k of each job in pl from 1 to the length of pl;
2: Update TEST $t'_{ES}(v_k)$ and TLFT $t'_{LF}(v_k)$ for $\forall v_k \in pl$;
3: **if** $\sum_{v_k \in pl} t_{k,1} > pl.lft - pl.est$ **then**
4: $t_1^S \leftarrow pl.est$;
5: **for** $v_k \in pl$ **do**
6: **if** $k > 1$ **then**
7: $t_k^S \leftarrow \max\{t_{k-1}^F, t'_{ES}(v_k)\}$;
8: $t_k^F \leftarrow t_k^S + t_{k,1}$; $L_k \leftarrow L_{k,1}$;
9: **if** $t_k^F > t'_{LF}(v_k)$ **then**
10: **return** False.
11: **return** False.
12: Use Algorithm 10 to calculate the execution option index $optIdx[k]$, AST t_k^S and AFT t_k^F for each job $v_k \in pl$ based on $\{tet_k | v_k \in pl\}$;

13: **for** $v_{k+1} \in pl$ **do**

14: **if** $t_k^F > t'_{LF}(v_k)$ or $t_k^F < t'_{ES}(v_{k+1})$ **then**

15: Let $pl(1,k)$ be the sub-pipeline of pl from its 1st to the k-th job;

16: $pl(1,k).est \leftarrow pl.est$;

17: **if** $t_k^F > t'_{LF}(v_k)$ **then**

18: $pl(1,k).lft \leftarrow t'_{LF}(v_k)$;

19: **else**

20: $pl(1,k).lft \leftarrow \min\{t'_{ES}(v_{k+1}), t'_{LF}(v_k)\}$

21: Clear $optIdx[k]$, t_k^S and t_k^F for each job v_k in pl;

22: **return** $PM(pl(1,k), tet)$;

23: Map v_k from t_k^S to t_k^F according to the $optIdx[k]$-th option in tet_k;

24: **return** True.

would not be cancelled because the CP with the earliest LFT is selected and mapped in each iteration.

In pipeline mapping in Algorithm 9, due to the homogeneity of tasks in a job, we map all the tasks in the same job onto VM instances in a single class, hence using Algorithm 10 to balance the trade-off between execution time and expense for each job on a pipeline (Line 12). When the jobs in a pipeline are mapped in their execution order, we check if everyone respects precedence constraints, and remap a pipeline with the updated LFT and length, if needed (Lines 13-22).

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

4.5 Performance Evaluation

4.5.1 Simulation Settings

We conduct simulations to evaluate our mapping model, referred to as Job-VMC, which maps each job with adaptive task partitioning onto a set of VM instances of types in the same class, in comparison with two existing mapping models: one mapping each task onto a VM instance, referred to as Task-VMI [71], and the other mapping each job with a preset number of tasks onto a set of VM instances of the same VM type, referred to as Job-VMT [73]. Then, we conduct simulations to evaluate the performance and financial cost of BAWM in our mapping model in comparison with existing algorithms: global-greedy-budget (GGB) in [71], gradual refinement (GR) in [71], and critical-path-greedy (CPG) in [73].

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

We conduct the simulation on five classes of VM types for different usage purposes, consisting of 20 VM types as tabulated in Table 4.4, based on real-life VM types in Amazon EC2. The parameters ϵ_1 in Algorithm 1 and ϵ_2 in Algorithm 10 are set to 0.05 and 0.01, respectively. By default, each data point denotes the average of 1000 runs with standard deviations; the workflow size and the number of VM types are set to be 100 jobs and 20, respectively; the maximum number L' of tasks for each job is randomly selected between 30 and 50; the budget factor β, which

Table 4.4 Specifications for Virtual Machine Types

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

is a factor determining the actual budget B based on a budget range $[e_{\min}, e_{\max}]$ as $B = e_{\min} + \beta(e_{\max} - e_{\min})$, is set to 0.3.

In each simulation, the mapping success rate (MSR) is defined as the ratio of the number of workflow requests (i.e., pairs of a workflow instance and a budget), each of which has a feasible mapping scheme, to the total number of workflow requests in a

Table 4.5 Problem Sizes

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

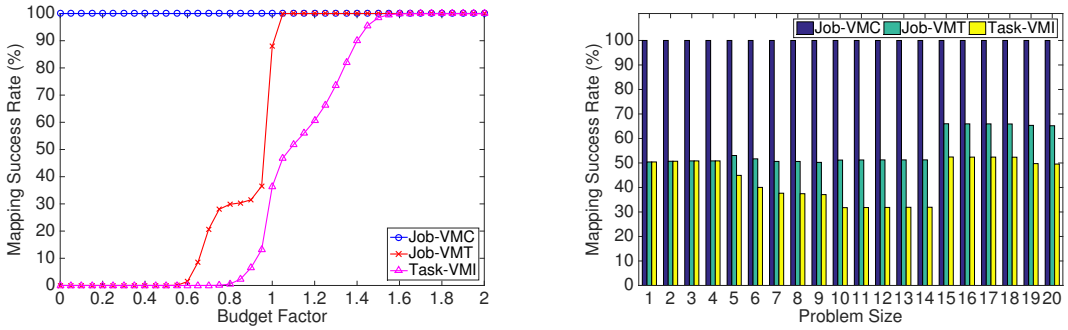
mapping model. The performance improvement, i.e., makespan reduction, over other algorithms in comparison is defined as

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

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

4.5.2 Mapping Success Rate for Mapping Models

We first examine the performance of Job-VMC, Job-VMT, and Task-VMI in terms of MSR with various problem sizes under different budget constraints. We consider 20 different problem sizes from small to large scales, indexed from 1 to 20, as tabulated in Table 5.6. Each problem size is defined as a triple of the number $|V|$ of jobs per workflow, the maximum number L' of tasks per job, and the number $|Y|$ of VM types.



(a) (b)

Figure 4.2 Mapping success rate: (a) in 2,000,000 instances (20 different problem sizes × 100,000 random workflow instances) for each budget factor; (b) in 2,100,000 instances (21 different budget factors × 100,000 random workflow instances) for each problem size.

The budget factor on the budget range $[e_{\min}, e_{\max}]$ calculated based on the Job-VMC mapping model varies from 0 to 2 in the evaluation. The MSR of these mapping models with different budget factors and problem sizes are plotted in Figure 4.2(a) and Figure 4.2(b), respectively. Figure 4.2(a) shows that Job-VMC achieves 100% MSR when $\beta \geq 0$, while Job-VMT and Task-VMI do not achieve 100% MSR until β reaches 1.1 and 1.7, respectively. Figure 4.2(b) shows that with different problem sizes, the MSR of BAWM is 100%, while the MSRs of Job-VMT and Task-VMI are only between 50.3% and 66.0%, and between 31.8% and 52.4%, respectively.

4.5.3 Makespan for Mapping Algorithms

Problem Size and Budget Scale We evaluate the performance of GGB, GR, CPG, and BAWM in the Job-VMC mapping model in terms of makespan and remaining budget with problem sizes from index 1 to 20 under budget constraints from 0 to 1 at an interval of 0.05. We plot the average makespan reduction and financial improvement of BAWM over GGB, GR, and CPG with different budget factors and problem sizes in Figures 4.3-4.8, respectively. These measurements show that BAWM reduces makespan by up to 30.8%, 49.1%, and 20.8%, and by 20.5%,

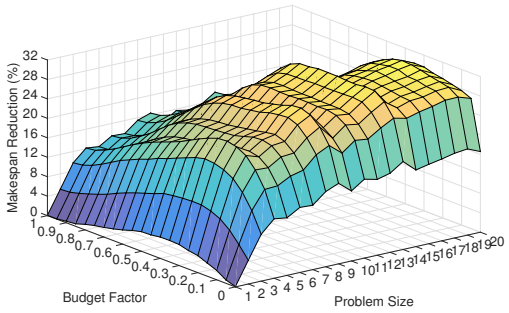


Figure 4.3 The average performance improvement of BAWM over GGB in 400 instances for each budget factor and each problem size.

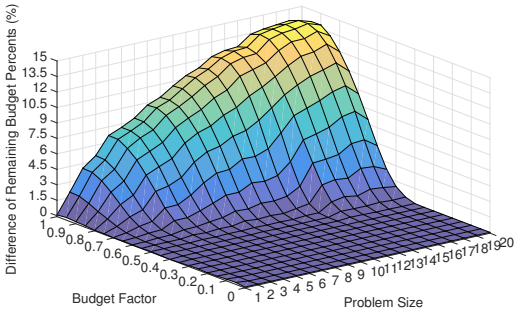


Figure 4.4 The average financial improvement of BAWM over GGB in 400 instances for each budget factor and each problem size.

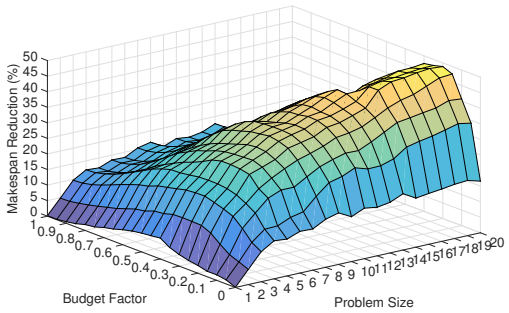


Figure 4.5 The average performance improvement of BAWM over GR in 400 instances for each budget factor and each problem size.

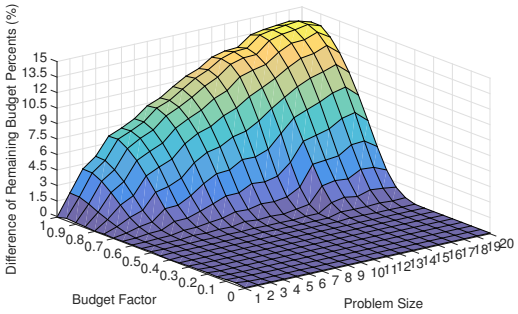


Figure 4.6 The average financial improvement of BAWM over GR in 400 instances for each budget factor and each problem size.

24.3%, and 8.4% on average in comparison with GGB, GR and CPG, respectively. The performance optimization of GGB and GR is limited by the bounds of each stage, and thus is inferior to that of BAWM. CPG, recursively improving the performance of the CP, ignores the impact of jobs in the non-critical paths on the budget usage, and thus causes unnecessary financial waste and limits its performance improvement. When the budget reaches $e_{\min}$ (or $e_{\max}$), all the jobs run the slowest (or fastest), and CPG and BAWM obtain the optimal makespan in polynomial time, so there is no performance improvement of BAWM over CPG in both scenarios. Furthermore,

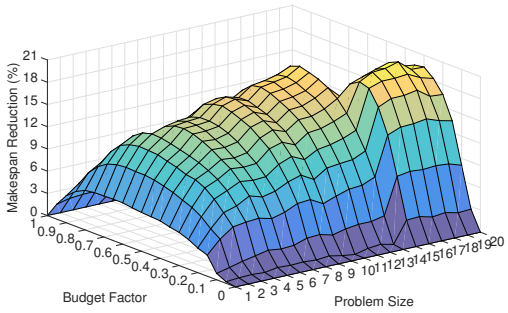


Figure 4.7 The average performance improvement of BAWM over CPG in 400 instances for each budget factor and each problem size.

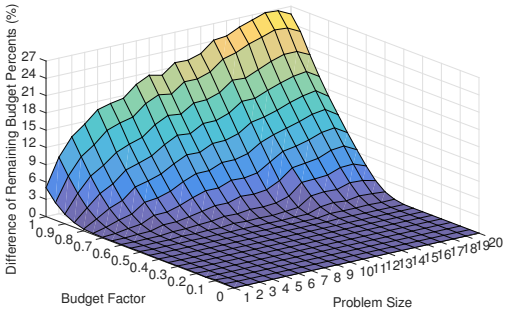


Figure 4.8 The average financial improvement of BAWM over CPG in 400 instances for each budget factor and each problem size.

when BAWM performs other three algorithms in terms of makespan, it increases the percentage of remaining budget by up to 14.3%, 14.3%, and 26.9%, and by 2.9%, 2.9%, and 4.3% on average in comparison with GGB, GR and CPG, respectively. With tight budgets, all these algorithms exhaust budgets to seek for the least end-to-end delay; with loose budgets, since all the jobs do not need to run the fastest to achieve the minimum makespan, BAWM save a significant amount of financial cost.

Workflow Size We execute GGB, GR, CPG, and BAWM in the Job-VMC model under different workflow sizes for scalability evaluation. We plot the makespan reduction in Figure 4.9, where we observe that as the number of jobs increases, BAWM improves the performance by 4.7% to 31.2%, 8.8% to 48.0%, and 8.3% to 20.4% in comparison with GGB, GR, and CPG, respectively, hence exhibiting a satisfactory scalability property with respect to the workflow size.

The Number of Virtual Machine Types We also run these algorithms in the Job-VMC model under different numbers of VM types. The set of available VM types are selected in the order of VM types listed in Table 4.4 every time. The makespan reduction of BAWM over GGB, GR, and CPG is plotted in Figure 4.10, which shows

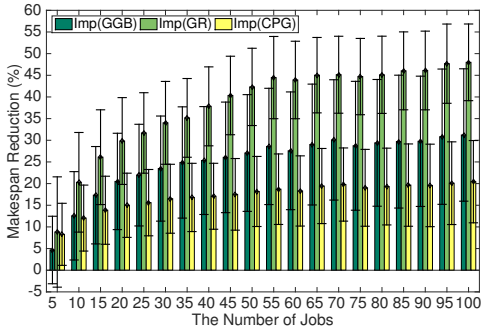


Figure 4.9 The performance improvement of BAWM in 1000 instances for each workflow size.

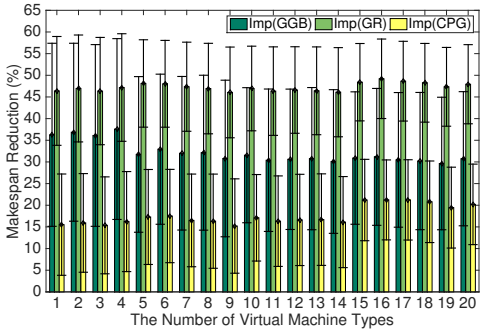


Figure 4.10 The performance improvement of BAWM in 1000 instances for each number of VM types.

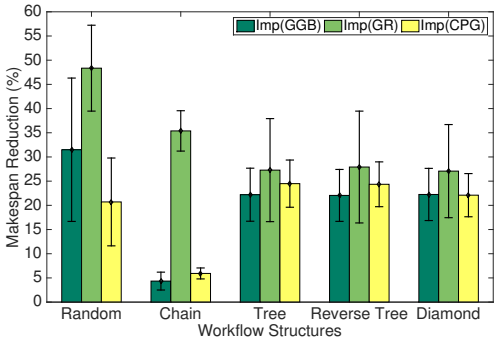


Figure 4.11 The performance improvement of BAWM in 1000 instances for each workflow structure.

that BAWM improves the performance by 29.6% to 37.6%, 46.0% to 49.2%, and 15.2% to 21.2% in comparison to GGB, GR, and CPG, respectively.

Workflow Structure We evaluate the performance of these algorithms in the Job-VMC model with different workflow structures, including a random shape, a chain, a tree, a reverse tree, and a diamond. The makespan reduction is plotted in Figure 4.11, which shows that BAWM improves the performance by 31.5%, 4.3%, 22.2%, 22.1% and 22.2%, by 48.4%, 35.4%, 27.3%, 27.9% and 27.1%, and by 20.7%, 5.9%, 24.5%, 24.4% and 22.1% in comparison with GGB, GR, and CPG for five different workflow structures, respectively. Although there is less room for performance improvement

in such a simple workflow structure as pipeline, BAWM still achieves considerable performance improvement as a result of incorporating Algorithm 1 for the first special case.

4.6 Conclusion

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

CHAPTER 5

ENERGY-EFFICIENT STATIC MAPPING OF MAPREDUCE WORKFLOWS IN SHARED CLUSTERS

This chapter is organized as follows. Section 5.1 introduces the background and significance of energy efficiency of MapReduce workflows. Section 5.2 formulates a MapReduce workflow mapping problem, and provides its strong NP-hardness proof. We prove a special case to be weakly NP-complete and design an FPTAS for it in Section 5.3, and design a heuristic for the generalized problem in Section 5.4. Section 5.5 evaluates the performance and Section 5.6 concludes our work.

5.1 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 research efforts on green computing were focused on a batch of independent MapReduce jobs in Hadoop systems or traditional workflows comprised of serial/rigid programs. Energy efficiency of large-scale workflows in big data systems such as Hadoop still remains largely unexplored.

Modern computing systems achieve energy saving mainly through two types of techniques, i.e., i) task consolidation to reduce static energy consumption (SEC) by turning off idle servers [17, 23, 12, 20], and ii) load balancing to reduce DEC through DVFS [37, 50, 46, 49, 80, 79], or a combination of both. However, these green computing techniques are not sufficient to address the energy efficiency issue of

big data workflows, because i) frequently switching on and off a server may reduce its lifespan or cause unnecessary peaks of power consumption, and ii) DVFS may not be always available on all servers in a cluster. According to the work by Chen *et al.* on the analysis of the impact of MapReduce operating parameters on energy efficiency under different workloads in Hadoop clusters [21], energy efficiency is equivalent to time efficiency if the amount of work accomplished per unit time is proportional to the amount of consumed resources. Following this line of research, 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.

In this chapter, 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 [25]. In our problem, it is challenging to balance the trade-off between energy cost and execution time of each component job to determine their respective completion time in MapReduce workflows, regardless of several previous efforts in traditional workflows, such as the partial critical path and minimum dependency methods in [10, 83].

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 an 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 DAG-structured workflow on a heterogeneous cluster. This heuristic iteratively selects the longest chain of unmapped jobs from the workflow

and applies our FPTAS to the selected pipeline while taking machine heterogeneity into consideration.

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

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

5.2 Problem Formulation

5.2.1 Cost Models

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

Table 5.1 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

Workflow Model We consider a user request in the form of a workflow $f(G, d)$, which specifies a workflow structure G and a deadline d. The workflow structure is

defined as a DAG $G(V, A)$, where each vertex $v_j \in V$ represents a component job, and each directed edge $a_{j,j'} \in A$ denotes an execution dependency, i.e., the actual finish time (AFT) t_j^{AF} of job v_j must not be later than the actual start time (AST) $t_{j'}^{AS}$ of job $v_{j'}$. The completion time of the workflow is denoted as t^C. We consider the map and reduce phases of each MapReduce job as two component jobs connected via an execution dependency edge.

MapReduce Model We consider a MapReduce job v_j running a set of parallel map (or reduce) tasks, each of which requires a memory of size o_j and spends a percentage $\mu_{i,j}$ of time executing CPU-bound instructions on a CPU core of machine m_i and a percentage $(1 - \mu_{i,j})$ of time executing I/O-bound instructions on machine m_i. In job v_j, generally, as the number K_j of parallel tasks increases, the workload $w_{j,k}(K_j)$ of each task $s_{j,k}$ decreases and the total workload $w_j(K_j) = K_j \cdot w_{j,k}(K_j)$ of all tasks increases. However, the maximum number K'_j of tasks that can be executed in parallel without performance degradation is limited by the cluster capacity, e.g., $K'_j \leq \sum_{m_i \in M} \min\{N_i, \lfloor o_i/o_j \rfloor\}$. Note that a serial program can be considered as a special case of a MapReduce job with $K'_j = 1$. The execution time of task $s_{j,k}$ on machine m_i is $t_{i,j,k} = w_{j,k}(K_j)/(\mu_{i,j} \cdot p_i)$. Estimating the execution time of a task on any service is an important issue. Many techniques have been proposed such as code analysis, analytical benchmarking/code profiling, and statistical prediction [61, 66], which are beyond the scope of this chapter.

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

Energy Model The DEC of a workflow in a cluster is

$$E = \sum\nolimits_{m_i \in M} \{P_i \sum\nolimits_{v_j \in V} [\mu_{i,j} \int_0^{t^C} n_{i,j}(t)dt]\},$$

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 [22].

Mapping Function We define a workflow mapping function as

$$\mathfrak{M} : \{s_k(v_j) \xrightarrow{[t^S_{j,k},\ t^E_{j,k}]} m_i, \forall v_j \in V, \exists m_i \in M, \exists [t^S_{j,k}, t^F_{j,k}] \subset T\},$$

which denotes that the k-th task of the j-th job is mapped onto the i-th machine from time $t^S_{j,k}$ to time $t^E_{j,k}$. The domain of this mapping function spans across all possible combinations of a set V of component jobs of the workflow, a set M of machines, and a time period T of workflow execution.

5.2.2 Problem Definition

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

Definition 5. ***EEWM****: Given a cluster $\{m_i(N_i, p_i, o_i, P_i)\}$ of machines with an available resource-time table $\{N^A_i(t), o^A_i(t)\}$, 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 $\mathfrak{M} : (V, M, T) \rightarrow \{s_k(v_j) \xrightarrow{[t^S_{j,k},\ t^E_{j,k}]} m_i\}$ to minimize the dynamic energy consumption:*

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

subject to the following time and resource constraints:

$$t^C \leq d,$$

$$t_j^{AF} \leq t_{j'}^{AS}, \forall a_{j,j'} \in A,$$

$$n_i(t) \leq N_i^A(t), \forall m_i \in M,$$

$$o_i(t) \leq o_i^A(t), \forall m_i \in M.$$

5.2.3 Complexity Analysis

We first consider a special case of EEWM with a sufficiently large upper bound on dynamic energy consumption as follows: given five machines with sufficient memory and a single CPU core of speed p and DPC P_i at full utilization, and J independent serial jobs $\{v_j\}$ with CPU-burst workload w_j, does there exist a feasible non-preemptive scheduling scheme such that the makespan is no more than d? This special case has been proved to be strongly NP-hard in [25], so is the general EEWM problem, which, with a polynomially bounded objective function, has no FPTAS unless $P = NP$ [69].

5.3 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 6. ***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?

5.3.1 Complexity Analysis

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

Definition 7. ***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 2. *EEWM-PHO is NP-complete.*

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

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

Then, if job v_j only has one task, its execution time is $t_j(1) = w_j(1)/(\mu_j p) = h_{j,1}$, and its DEC is $E_j(1) = t_j(1)\mu_j P = B_j - b_{j,1}$. If job v_j has two tasks, the

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

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

5.3.2 Approximation Algorithm

We prove that EEWM-PHO is weakly NP-complete and design an FPTAS as shown in Algorithm 10 by reducing this problem to the weakly NP-complete restricted shortest path (RSP) problem, which can then be solved using an FPTAS proposed in [28].

Given an instance of EEWM-PHO, we construct an instance of RSP according to the pipeline-structured workflow as follows. As illustrated in Figure 5.1, the network graph $\mathbb{G}$ consists of $\mathbb{V} = \{v_{j,k} | j = 1, \ldots, J, k = 1, \ldots, K'_j\} \cup \{u_0, u_j | j = 1, \ldots, J\}$ with a source u_0 and a destination u_J, and $\mathbb{E} = \{e_{2j-1,k}, e_{2j,k} | j = 1, \ldots, J, k = 1, \ldots, K'_j\}$, where $e_{2j-1,k} = (u_{j-1}, v_{j,k})$ and $e_{2j,k} = (v_{j,k}, u_j)$. Then, we calculate the execution time of job v_j with k tasks as $t_j(k) = w_j(k)/(k \cdot p \cdot \mu_j)$, and accordingly its DEC as $E_j(k) = k \cdot P \cdot \mu_j \cdot t_j(k)$. Subsequently, we assign the cost $c(e)$ and delay $l(e)$ of each edge $e \in \mathbb{E}$ as $c(e_{2j-1,k}) = E_j(k)$, $l(e_{2j-1,k}) = t_j(k)$, and $c(e_{2j,k}) = l(e_{2j,k}) = 0$, and set the delay constraint on a path from u_0 to u_J to be d. As a result, the minimum cost in RSP is exactly the minimum DEC in EEWM-PHO, and if $v_{j,k}$ is on the solution path to RSP, the j-th job has k tasks. Based on Theorem 2 and the above reduction, we have

Theorem 3. *EEWM-PHO is weakly NP-complete.*

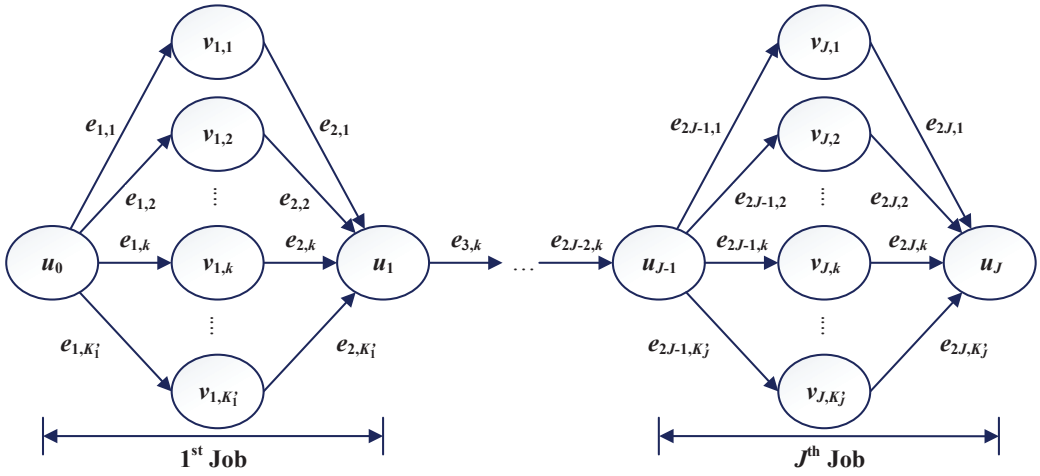


Figure 5.1 A constructed network corresponding to a workflow with a pipeline structure.

Algorithm 10: 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 Figure 5.1, and assign energy cost $E_j(k)$ and delay $t_j(k)$ to edge $e_{2j-1,k}$ and zero cost and zero delay to edge $e_{2j,k}$;

2: Use FPTAS in [28] 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.

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 [28] is used to solve RSP in acyclic graphs. Thanks to the special topology in Figure 5.1, the time complexity is further reduced to $O(JK'(\log K' + 1/\epsilon))$.

5.4 Algorithm 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).

5.4.1 An Overview of BAWMEE

The key idea of BAWMEE is to partition a DAG into a set of pipelines and then repeatedly employ Algorithm 10 with near optimality and low time complexity to achieve energy-efficient mapping of each pipeline.

In BAWMEE, each workflow mapping consists of two components: iterative critical path (CP) selection and pipeline mapping. A CP is the longest execution path in a workflow, which can be calculated in linear time. The algorithm starts with computing an initial CP according to the average execution time of each job running in serial on all the machines, followed by a pipeline mapping process. Then, it iteratively computes a CP with the earliest last finish time (LFT) from the remaining unmapped workflow branches based on the same average execution time of a job as above and performs a pipeline mapping of the computed CP until there are no branches left.

In pipeline mapping, we consider two extreme scenarios: resource/time sufficiency and resource/time insufficiency. In the former case, we only need to focus on energy efficiency, while in the latter case, it may be unlikely to meet the performance requirement. Therefore, we design one algorithm for each of these two scenarios: a heuristic for energy-efficient pipeline mapping (EEPM) under a deadline constraint in Algorithm 12, which calls Algorithm 10, and a heuristic for minimum delay pipeline mapping (MDPM) with energy awareness in Algorithm 13. If Algorithm 12 fails to find a feasible mapping scheme due to limited resources, we resort to Algorithm 13. 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 Algorithm 10 to balance the trade-off

Table 5.2 Time-Energy Table Tbl_j of Job v_j

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

between execution time and DEC (directly associated with total workload) for each job on a pipeline. In MDPM, we search for a good task partitioning to minimize the end time of each job through a limited number of tries by reducing the possible number of tasks in each job v_j from $\{1, 2, 3, \ldots, K'_j\}$ to $\{1, 2, 2^2, \ldots, 2^{\lfloor \log K'_j \rfloor}\} \cup \{K'_j\}$.

5.4.2 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 5.2, 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 Algorithm 11, BAWMEE first builds a time-energy table for each job by calling $buildTET()$ (in Line 1). If the workflow cannot meet its deadline with each job running the fastest, BAWMEE performs energy-aware job mapping (EAJM) with minimum finish time for each job in a topologically sorted order by calling $simplyMap()$ (in Line 2). Otherwise, BAWMEE employs iterative CP selection to find a CP with the earliest LFT from unmapped jobs (in Line 8), and performs EEPM or MDPM (if EEPM fails) for the selected CP (in Lines 9-10), where EEPM and MDPM are described later in Algorithms 12 and 13, respectively. If there is any job that cannot be mapped in MDPM, we cancel the mapping of its downstream jobs (in Lines 11-14). If it is the last job of the workflow, we perform EAJM with minimum finish time (in Lines 15-16).

In Algorithm 12 of EEPM, we reset the EST for the input pipeline according to the earliest time such that enough resources are made available to the first job (in Lines 2-3). If the pipeline cannot meet its LFT with each job running the fastest, we exit EEPM (in Lines 4-5); otherwise, the mapping of a pipeline with its EST and LFT is converted into the RSP problem with a relaxed resource limit (in Line 6). Accordingly, we calculate the number of tasks, the sub-cluster, and the start/finish time for each job using Algorithm 10 (in Line 7). Then, we check if the start and

Algorithm 11: BAWMEE

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

1: $Tbl \leftarrow buildTET(V, \{C_l\})$;

2: **if** $simplyMap(f, R(\{C_l\}), Tbl)$ =True **then**

3: **return** .

4: $t_j^{LF} \leftarrow +\infty$ for $\forall v_j \in f$; $t_J^{LF} \leftarrow d$ for the end job v_J in f;

5: Calculate the average execution time $\bar{t}_j$ of each job v_j running in serial on all the machines;

6: $G' \leftarrow G$;

7: **while** $\exists$ an unmapped job $\in V$ **do**

8: Find the critical path cp ending at a job v with the earliest LFT in G' according to $\{\bar{t}_j | v_j \in G'\}$;

9: **if** $EEPM(cp, R(\{C_l\}), Tbl)$ =False **then**

10: $v \leftarrow MDPM(cp, R(\{C_l\}))$;

11: **if** $v \neq Null$ **then**

12: $D \leftarrow$ {all the downstream jobs of v in $G - G'$};

13: **if** $D \neq \emptyset$ **then**

14: Cancel the mapping of each job $v' \in D$, and add v' and its associated precedence constraints to G';

15: **if** v is the last job of f **then**

16: $EAJM(v, R(\{C_l\}))$;

17: $G' \leftarrow G' - \{v_j \in cp | v_j$ is mapped$\}$;

finish time of each job are between its TEST and TLFT in their execution order (in Lines 8-9). If there exists a job that violates the precedence constraint, we divide

the pipeline at this job, and use Algorithm 12 to compute the mapping of the upstream sub-pipeline with an updated LFT constraint (in Lines 10-15). We repeat this process until we find a sub-pipeline whose mapping meets all precedence constraints. If the cluster is able to provide each job in this sub-pipeline with enough computing resources based on the mapping result of Algorithm 10, we proceed with this mapping (in Lines 16-18); otherwise, we fail to find an EEPM and thus exit (in Line 19). In this case, BAWMEE would proceed to search for an MDPM.

In Algorithm 13 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 Algorithm 14 of EAJM with the minimum finish time under resource constraints, we exponentially relax the limit on the maximum number of tasks in a job to make a tradeoff between the optimality and the time complexity of EAJM.

Since the calculation of the earliest possible start time of the first job in EEPM takes time of $O(M'H)$ and the pipeline mapping in EEPM takes time of $O(JK'L[\log(K'L) + 1/\epsilon])$, the time complexity of EEPM is $O(J^2K'L[\log(K'L) + 1/\epsilon] + M'H)$. Since EAJM takes time of $O(M'HK' \log K')$, the time complexity

Algorithm 12: 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

1: Label the index j of each job in pl from 1 to the length of pl;

2: Calculate the earliest possible start time of the first job in pl on any machine as est according to $R(\{C_l\})$;

3: $pl.est \leftarrow \max\{est, pl.est\}$;

4: **if** $\sum_{v_j \in pl} t_j(K_{j,1}, C_{j,1}) > pl.lft - pl.est$ **then**

5: **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 Algorithm 10 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;

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

5.4.3 Numerical Examples

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

Algorithm 13: 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 14: 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}.

The first example considers an idle cluster $M = C_1 \cup C_2$ consisting of 4 single-core machines, where $C_1 = \{m_1, m_2\}$ and $C_2 = \{m_3, m_4\}$, and receives a workflow f comprised of homogeneous jobs organized in Figure 5.2 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 5.3.

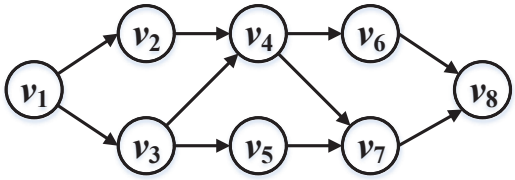


Figure 5.2 An example of a workflow structure G.

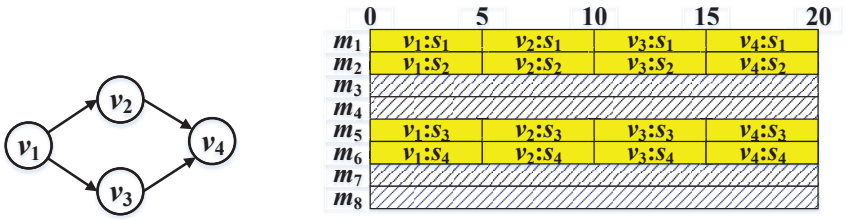
Table 5.3 Time-Energy Table in Example 1

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

(a) (b)

Figure 5.3 Workflow mapping in example 1: (a) BAWMEE; (b) Optimal.

BAWMEE first builds a TET for each job on the right side of Table 5.3. 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,



(a) (b)

Figure 5.4 Example 2: (a) workflow structure; (b) workflow mapping.

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 $G - \{v_1, v_2, v_3, v_4, v_5, v_6, v_8\}$, 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 Figure 5.3(a), and its DEC is 45 units. The optimal mapping is shown in Figure 5.3(b), and the minimum DEC is 44 units.

The second example considers a cluster $M = C_1 \cup C_2$ consisting of 8 single-core machines, where $C_1 = \{m_1, m_2, m_3, m_4\}$ and $C_2 = \{m_5, m_6, m_7, m_8\}$, and m_3, m_4, m_7 and m_8 are busy and occupied by previous workflows. A user request specifies a workflow f comprised of homogeneous jobs organized in Figure 5.4(a) with a

Table 5.4 Time and Energy per Job in Example 2

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

deadline of 15 time units. The execution time and DEC of a job with a different task partitioning on a different sub-cluster are calculated and listed in Table 5.4. A pipeline $\{v_1, v_2, v_4\}$ is selected as the initial CP. EEPM intends to perform pipeline mapping with EST of 0 and LFT of 15 by partitioning each job of v_1, v_2 and v_4 into four tasks and mapping them onto C_1. However, C_1 does not have enough resources to support this mapping. Due to the failure of EEPM, MDPM attempts to partition each job into one, two, and four tasks to search for the minimum completion time of each job one by one. As a result, v_1, v_2 and v_4 are all partitioned into four tasks, and mapped onto m_1, m_2, m_5 and m_6 from 0 to 5, from 5 to 10, and from 10 to 15, respectively. Then, the second pipeline $\{v_3\}$ with EST of 5 and LFT of 10 is selected as the CP in $G - \{v_1, v_2, v_4\}$, but fails to be mapped during time window $[5, 10]$ by EEPM and MDPM due to insufficient resources. Hence, BAWMEE cancels the mapping of the downstream mapped job $\{v_4\}$ of v_3, which is the first job that fails to be mapped before its TLFT of 10 by MDPM. Subsequently, the third pipeline $\{v_3, v_4\}$ with EST of 10 and LFT of 15 is selected as the CP in $G - \{v_1, v_2\}$, and fails to be mapped by EEPM. Thus, MDPM partitions v_3 into four tasks and maps them onto M from 10 to 15, but does nothing for v_4 due to missing its TLFT of 15. Finally, BAWMEE partitions the end job v_4 of the workflow into four tasks and maps them onto M from 15 to 20. Specifically, the mapping result of BAWMEE is shown in Figure 5.4(b), and its DEC is 176 units.

5.5 Performance Evaluation

We conduct experiments to illustrate the effect of task partitioning on job workload and energy consumption, and evaluate the performance of EEWM-PHO-FPTAS in a practical setting for the special case of a pipeline-structured workflow on a homogeneous cluster in comparison with the default and optimal workflow mapping schemes. For the generalized problem, we conduct simulations to evaluate the performance of BAWMEE in comparison with three existing algorithms adapted from different scenarios: i) EEDAW in Algorithm 16 adapted from a MapReduce job scheduling algorithm EEDAJ in Algorithm 15 (integrated with the algorithms in [22] and [23]) by extending the progress estimation of a MapReduce job to that of a workflow, ii) MinD+ED adapted from a workflow scheduling algorithm with serial jobs in [83] by fixing the number of tasks in each MapReduce job and replacing preemptive task scheduling with non-preemptive task scheduling, and iii) MinD+EEDAJ comprised of the MinD algorithm in [83] for determining the virtual deadline of each job in a workflow and EEDAJ for scheduling MapReduce jobs onto energy-efficient machines before their virtual deadlines. In these three existing algorithms, we preset the number of tasks in each MapReduce job to be the maximum number of tasks to illustrate the benefits brought forth by the adaptive task partitioning strategy in our algorithm.

5.5.1 Experiments

Although EEWM-PHO-FPTAS is designed for the special case of a pipeline-structured MapReduce workflow on a homogeneous cluster, it is the most important component of BAWMEE to solve the generalized problem.

Experimental Settings The testbed is the same as described in Subsection 3.2.1. On the cluster in our testbed, we also install Oozie 4.3 [3], a workflow engine that dispatches each component MapReduce job in a workflow with its respective

Algorithm 15: EEDAJ()

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

1: **while** $Q_J \neq \emptyset$ **do**

2: $v \leftarrow Q_J.top()$; // Q_J is a priority queue of all ready jobs. The priority of each job v is based on its deadline and execution progress, i.e., $rank(v) = d(v) - \sum_{s \in U(v)} \bar{t}(s)$, where $U(v)$ is a set of all unmapped tasks in v, and $\bar{t}(s)$ is the average execution time of task s on all machines.

3: **if** v is ready **then**

4: Select a task s from job v and estimate its expected finish time $d(s) = d(v) - [d(v) - t^{ES}(v)] \cdot (|U(v)| - 1)/|v|$, where $|v|$ is the number of tasks in v;

5: Map task s to minimize incremental energy consumption before $d(s)$ or to minimize finish time if the former fails;

6: **if** s is the last task in v **then**

7: Update the AFT of v and the EST of all its succeeding jobs;

8: $Q_J.dequeue()$;

9: **else**

10: Sleep for a period Δt; // $\Delta t = 6$ seconds

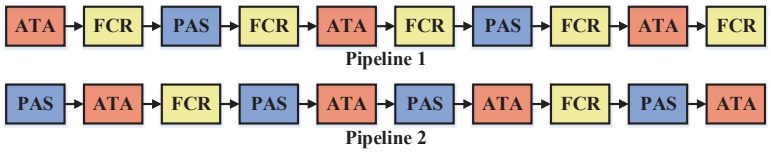


Figure 5.5 Pipeline-structured MapReduce workflows.

configuration once all its preceding jobs finish. We generate two pipeline-structured workflows, each comprised of 10 MapReduce jobs, as shown in Figure 5.5. These jobs

Algorithm 16: EEDAW()

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

1: **while** $Q_W \neq \emptyset$ **do**

2: $f \leftarrow Q_W.top()$; // Q_W is a workflow priority queue. The priority of each workflow f is based on its deadline $d(f)$ and execution progress, i.e., $rank(f) = d(f) - \sum_{s \in U(f)} \bar{t}(s)$, where $U(f)$ is a set of all unmapped tasks in f, and $\bar{t}(s)$ is the average execution time of task s on all machines.

3: $v \leftarrow Q_J(f).first()$; // The jobs in the job queue $Q_J(f)$ of workflow f follow a topological sorting.

4: Estimate the virtual deadline $d(v)$ of job v by $d(v) = t^{ES}(v) + [d(f) - t^{ES}(v)] \cdot \bar{t}(v)/[\bar{t}(v) + \sum_{v_j \in D(v)} \bar{t}(v_j)]$, where $\bar{t}(v) = \sum_{s \in v} \bar{t}(s)$.

5: **if** v is ready **then**

6: Select a task s from job v;

7: Map task s to minimize incremental energy consumption before $d(v)$ or to minimize finish time if the former fails;

8: **if** s is the last task in v **then**

9: Update the AFT of v and the EST of all its succeeding jobs;

10: $Q_J(f).dequeue()$;

11: **if** $Q_J(f) = \emptyset$ **then**

12: $Q_W.pop()$;

13: **else**

14: Sleep for a period Δt; // $\Delta t = 10$ minutes

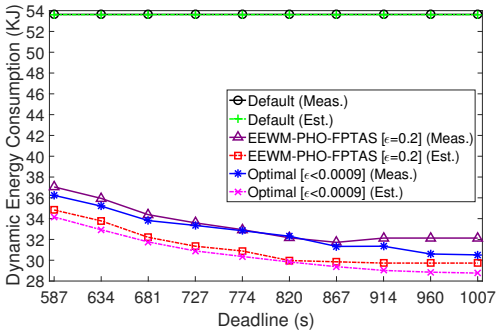


Figure 5.6 The DEC of Pipeline 1 under different deadline constraints.

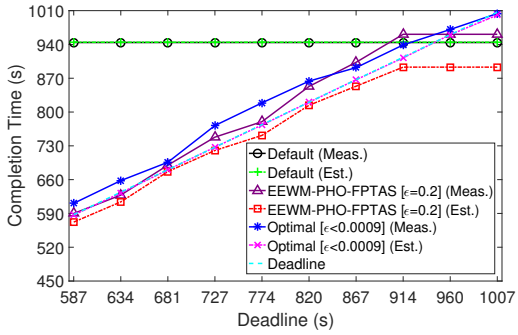


Figure 5.7 The completion time of Pipeline 1 under different deadline constraints.

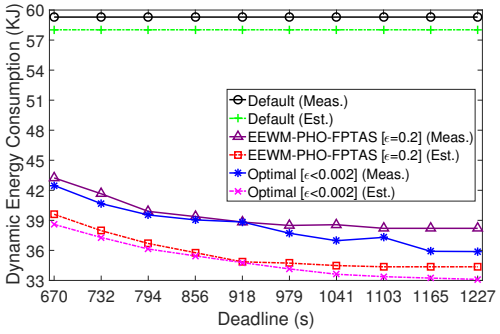


Figure 5.8 The DEC of Pipeline 2 under different deadline constraints.

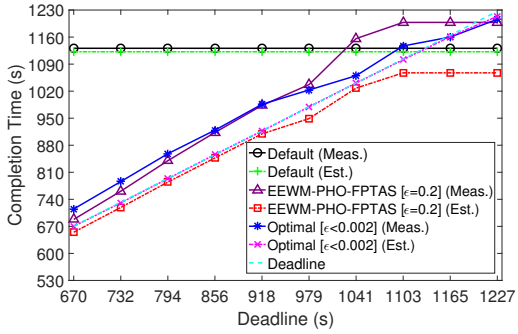


Figure 5.9 The completion time of Pipeline 2 under different deadline constraints.

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 the existing energy-efficient MapReduce workflow mapping algorithms do not adjust the number of mappers and reducers, their workflow mapping schemes in this special case are exactly the same and completely rely on the default settings in Hadoop, where the number of mappers is the input size divided by the split size of 128 MB, and the number of reducers is 1. Hence, we refer to the mapping scheme produced by these existing algorithms as the "default" scheme in this scenario.

Experimental Results To test the practical performance of EEWM-PHO-FPTAS, we conduct the workflow experiment on our homogeneous cluster, and plot in Figures 5.6-5.9 the analytical estimations and experimental measurements of the DEC and completion time based on the workflow mapping scheme produced by EEWM-PHO-FPTAS, as well as the default and optimal workflow mapping schemes under 10 different deadline constraints. The experimental measurements show that EEWM-PHO-FPTAS with $\epsilon = 0.2$ cuts down 27% to 40% DEC at the cost of up to 6% more computing time in comparison with the default mapping scheme, and consumes only 6% more dynamic energy in comparison with the optimal mapping scheme. Hence, these results clearly illustrate the performance superiority of EEWM-PHO-FPTAS over existing energy-efficient workflow mapping algorithms in practice. Furthermore, we observe that the differences between the analytical estimations and the experimental measurements are less than 8% for the first pipeline and 11% for the second pipeline, which indicates the accuracy of our cost models in describing the main characteristics of workflow execution on a real Hadoop cluster. These discrepancies are mainly caused by ignoring the impact of the number of mappers and reducers on the execution time and DEC of shuffling in MapReduce jobs, and the measurement errors on reduce tasks.

5.5.2 Simulation

Simulation Settings To further evaluate the performance of the proposed heuristic for the generalized problem of larger scales, we conduct extensive simulations in various scenarios. We first generate a series of random workflows as follows: (i) randomly select the length L of the critical path of a workflow (no less than 3) and divide the workflow into L levels, in each of which every job has the same length of the longest path from the start job; (ii) randomly select the number of jobs in each level except the first and last levels, in which there is only one job; (iii) for each

job, add an input edge from a randomly selected job in the immediately preceding level, if absent, and an output edge to a randomly selected job in its downstream level(s); (iv) randomly pick up two jobs in different levels and add a directed edge from the job in the upstream level to the job in the downstream level until we reach the given number of edges. The number of precedence constraints of the workflow is set to 1.5 times of the number of jobs, if possible. The maximum possible number of tasks for each job is randomly selected between 12 and 48. The workload of a job is randomly selected between 0.6×10^{12} and 21.6×10^{12} CPU cycles when running in serial. According to the performance model of moldable jobs, the workload $w(k)$ of a job with $k > 1$ tasks is randomly selected between $w(k-1)[1 + 0.2/(k-1)]$ and $w(k-1)[1 + 0.6/(k-1)]$. We calculate the sum t_1 of the average execution time of the serial jobs on the critical path and the sum t_2 of the average execution time of all serial jobs according to the CPU speeds of all types of machines, and randomly select a workflow deadline baseline from the time range $[t_1, t_2]$. The percentage of execution time for the CPU-bound instructions of a task in each job on each type of machine is randomly selected from 0.6 to 1 at an interval of 0.1. By default, the amount of memory to request from the scheduler for each map/reduce task is 1GB in Hadoop/YARN. Based on our empirical study, we randomly select the memory demand of a task in each job from a range between 0.5GB and 4GB at an interval of 0.5GB.

We evaluate these algorithms in a heterogeneous cluster consisting of machines with four different specifications listed in Table 5.5, based on four types of Intel processors. Each homogeneous sub-cluster has the same number of machines. Each scheduling simulation lasts for 3 days and is repeated for 20 times with different workflow instances, whose arrivals follow the Poisson distribution. In the performance evaluation, each data point represents the average of 20 runs with a standard deviation. We set parameter ϵ in BAWMEE to be 0.2 to balance between workflow

Table 5.5 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

energy consumption and algorithm execution time. According to Figures 5.6-5.9, when ϵ is set to be 0.2, the energy optimization performance is close to the optimal solution and BAWMEE is a polynomial-time solution. By default, the workflow size is randomly selected between 40 and 60 jobs; the cluster size and the average arrival interval of workflows are set to be 128 machines and 30 minutes, respectively; the deadline factor, which is a coefficient multiplied by the deadline baseline to determine the actual workflow deadline, is set to 0.15.

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

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

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

Table 5.6 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)

Simulation Results *Problem Size*: For performance evaluation, we consider 20 different problem sizes from small to large scales, indexed from 1 to 20 as tabulated in Table 5.6. Each problem size is defined as a quadruple $(|V|, |M|, 1/\lambda, T)$, where $1/\lambda$ is the average arrival interval of workflow requests in minutes, and T is the time period in unit of days for accepting workflow requests in each simulation. As the workflow size and arrival frequency increase from index 1 to 20, we increase the resources correspondingly to meet tight deadlines with factor 0.15. We plot the DECR, DMR, and URT of EEDAW, MinD+ED, MinD+EEDAJ, and BAWMEE in Figures 5.10-5.12, respectively, which show that BAWMEE saves 5.3% to 35.6%, 5.9% to 33.3%, and 6.3% to 34.5% DEC, and misses less deadlines in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively. Furthermore, the URT of BAWMEE is on the same order of magnitude as those of EEDAW, MinD+ED, and MinD+EEDAJ, and is less than 13 seconds even for problem index 20. We also plot the average number of tasks per job and average workload reduction of BAWMEE in Figure 5.13, which sheds light on the energy efficiency of BAWMEE.

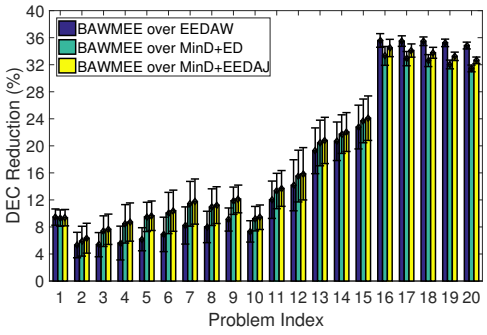


Figure 5.10 The DECR vs. problem sizes.

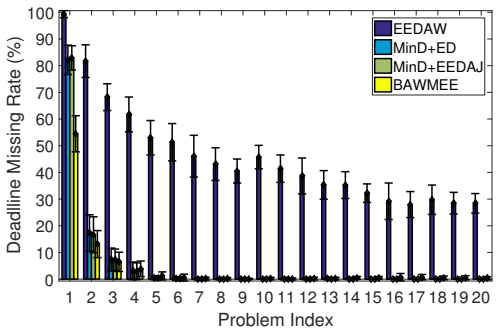


Figure 5.11 The DMR vs. problem sizes.

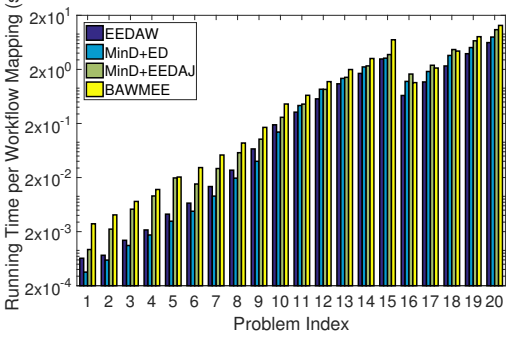


Figure 5.12 The URT vs. problem sizes.

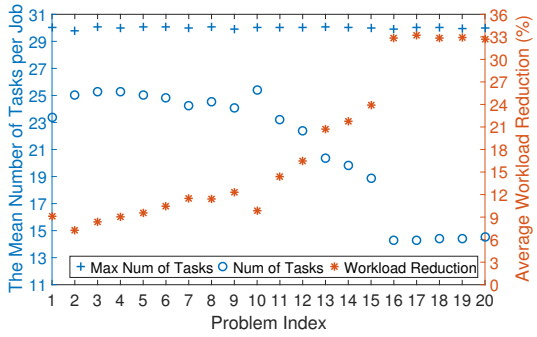


Figure 5.13 The adaptive task partitioning of BAWMEE vs. problem sizes.

We observe from all the problem indices in Figures 5.10 and 5.13 that on average a smaller number of tasks in each job would result in more reduced workload and thus more DEC reduction achieved by BAWMEE.

Deadline Constraint: We evaluate the performance of EEDAW, MinD+ED, MinD+EEDAJ, and BAWMEE in terms of DEC, DMR, and URT under different deadline constraints obtained from the deadline baseline multiplied by a factor from 0.05 to 1 with an interval of 0.05. The DEC, DMR, and URT of these algorithms are plotted in Figures 6.10-5.16, respectively. These measurements show that BAWMEE saves up to 23.7%, 27.5%, and 28.2% DEC as the deadline increases in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively, and reduces DMR from 99.9% to 93.0% with a deadline factor of 0.05 and from 83.3% to 25.9% with a

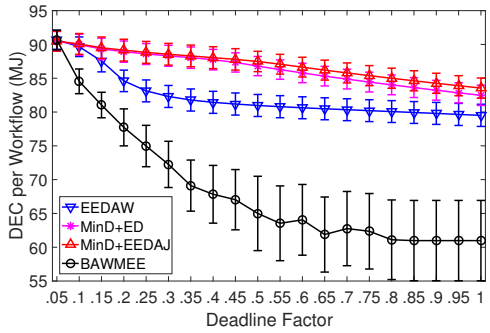


Figure 5.14 The DEC vs. deadlines.

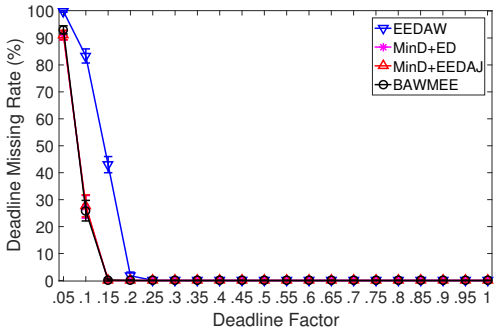


Figure 5.15 The DMR vs. deadlines.

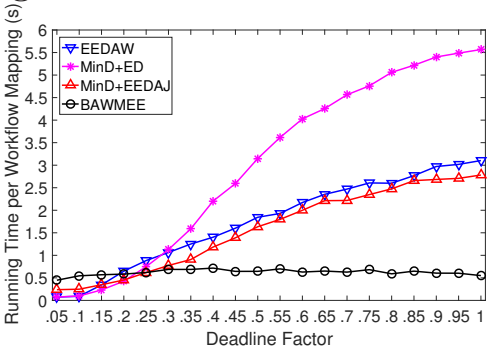


Figure 5.16 The URT vs. deadlines.

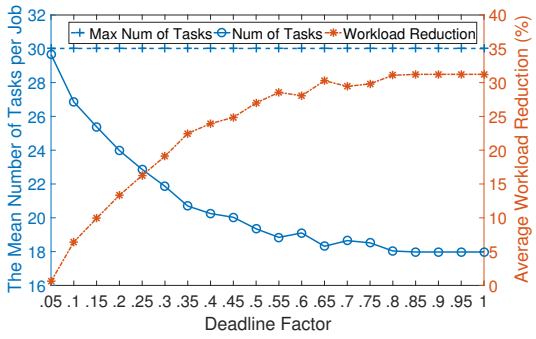


Figure 5.17 The adaptive task partitioning of BAWMEE vs. deadlines.

deadline factor of 0.1 compared to EEDAW. The DMR of BAWMEE is close to zero when the deadline factor is larger than 0.15, and is similar to those of MinD+ED and MinD+EEDAJ under various deadline constraints. Additionally, the URT of BAWMEE is less than 0.7 second and is 17.7% to 5.9, 9.8% to 6.0, and 45.6% to 5.1 times of those of EEDAW, MinD+ED, and MinD+EEDAJ, respectively. It is worth pointing out that as the deadline increases, the DEC and URT of BAWMEE decrease, because EEPM plays a more significant role than MDPM in BAWMEE. We plot the average number of tasks per job and the average workload reduction of BAWMEE under different deadline constraints in Figure 5.17, which clearly shows that BAWMEE reduces more workload overhead due to a decreased number of tasks as the deadline is relaxed, and explains why BAWMEE makes a better tradeoff between

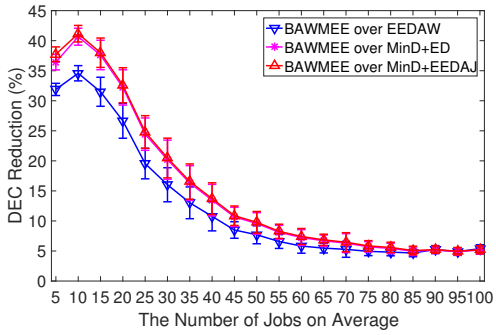


Figure 5.18 The DECR vs. workflow sizes.

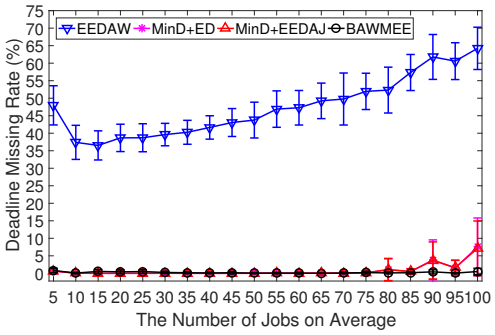


Figure 5.19 The DMR vs. workflow sizes.

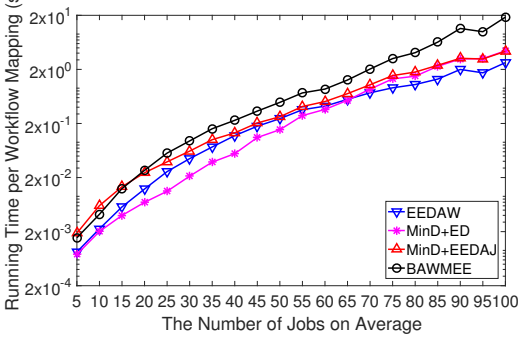


Figure 5.20 The URT vs. workflow sizes.

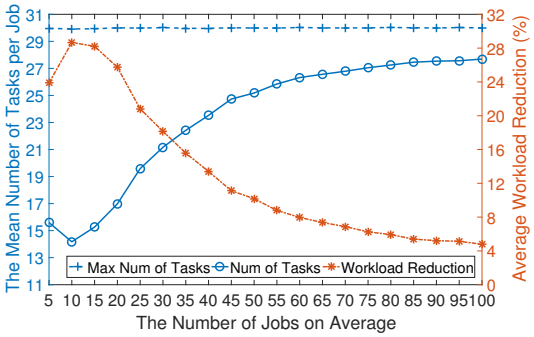


Figure 5.21 The adaptive task partitioning of BAWMEE vs. workflow sizes.

DEC and DMR than the other algorithms in comparison at an acceptable cost of running time.

Workflow Size: For scalability evaluation, we run these four algorithms under different average workflow sizes with 5 to 100 jobs per workflow at an interval of 5, where the maximum and minimum workflow sizes are 2 jobs more and less than the average workflow size, respectively. We plot the DECR, DMR, and URT of these algorithms in Figures 6.12-5.20, respectively, where we observe that BAWMEE with DMRs close to zero achieves an increasing DECR from 4.7% to 34.6%, from 4.9% to 40.7%, as well as from 5.0% to 41.2% in comparison with EEDAW, MinD+ED, and MinD+EEDAJ, respectively. For large workflow sizes with 50 to 100 jobs per workflow that impose high resource demands, BAWMEE achieves DECR only from 4.7% to

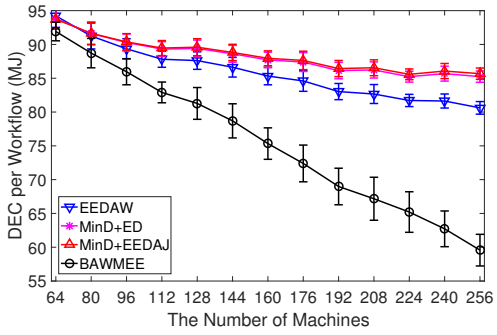


Figure 5.22 DEC vs. cluster sizes.

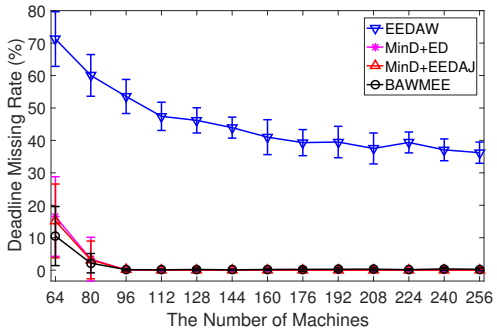


Figure 5.23 The DMR vs. cluster sizes.

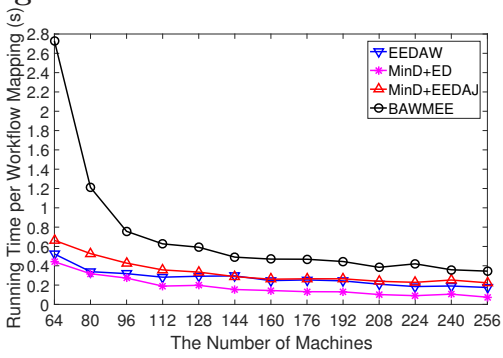


Figure 5.24 The URT vs. cluster sizes.

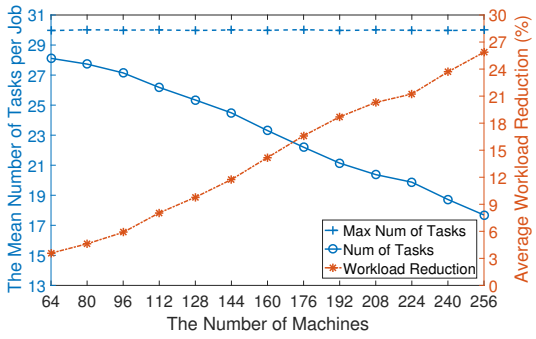


Figure 5.25 The adaptive task partitioning of BAWMEE vs. cluster sizes.

9.8%, because MDPM plays a more significant role than EEPM in BAWMEE, which is justified by the changes in the average number of tasks per job and the average workload reduction of BAWMEE plotted in Figure 5.21. The DMR of EEDAW experiences a slump under the medium workflow sizes because a higher accuracy could be achieved in the execution progress of a smaller workflow than a larger one, while a further increase in the workflow size may lead to a more severe shortage of computing resources. In addition, the URT of BAWMEE is comparable with those of EEDAW, MinD+ED, and MinD+EEDAJ.

Cluster Size: We run these four algorithms under different cluster sizes of 64 to 256 machines at a step of 16 for scalability test. The DEC, DMR, and URT of these algorithms are plotted in Figures 6.13-5.24, respectively, where we observe that as the number of machines increases, BAWMEE consumes 2.5% to 26.1%, 2.0% to

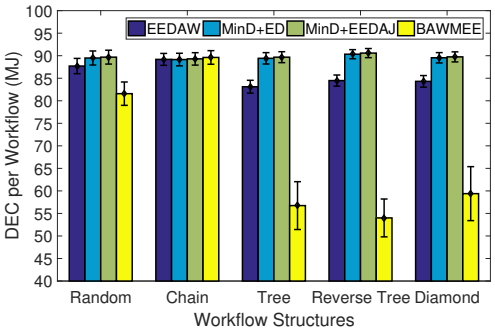


Figure 5.26 The DEC vs. workflow structures.

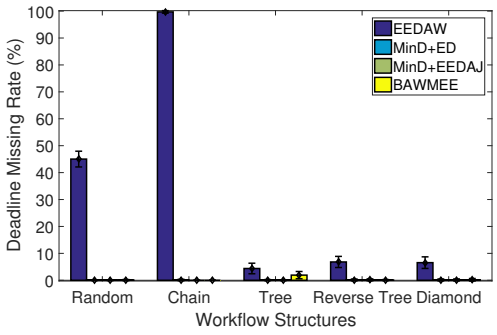


Figure 5.27 The DMR vs. workflow structures.

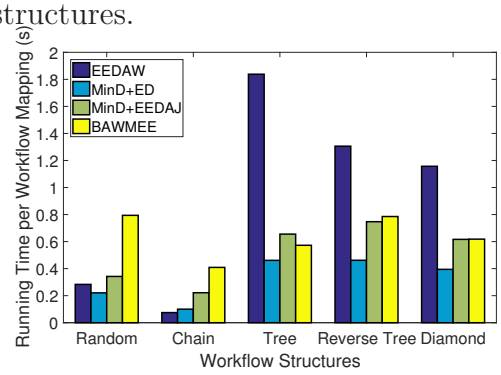


Figure 5.28 The URT vs. workflow structures.

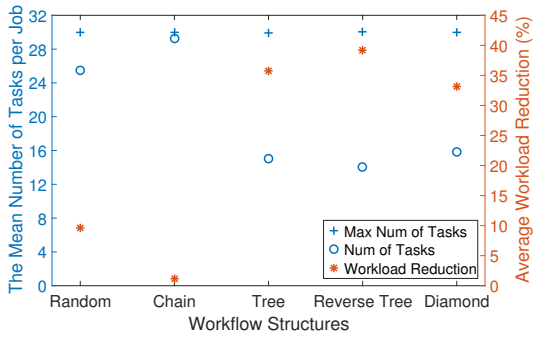


Figure 5.29 The adaptive task partitioning of BAWMEE vs. workflow structures.

30.1%, and 1.9% to 30.5% less DEC than EEDAW, MinD+ED, and MinD+EEDAJ, respectively, hence exhibiting a satisfactory scalability property with respect to the cluster size. Furthermore, the DMR of DAWMEE is only between 0.1% and 10.5% and is similar to those of MinD+ED and MinD+EEDAJ, while EEDAW misses 36.2% to 71.2% deadlines. The increase in the cluster size results in a relatively looser deadline and a more flexible workflow mapping, as a result of which, the DEC and DMR of these four algorithms decrease, and BAWMEE has more chances to save energy, which is consistent with the changes in the average number of tasks per job and the average workload reduction of BAWMEE plotted in Figure 5.25. Moreover, the URT of BAWMEE is less than 2.7 seconds and is comparable with those of EEDAW, MinD+ED, and MinD+EEDAJ.

Workflow Structure: We further investigate these four algorithms with various workflow structures, including a random shape, a chain, a tree, a reverse tree, and a diamond. The DEC, DMR, and URT are plotted in Figures 5.26-5.28, respectively, which show that BAWMEE reduces DEC by 6.9% to 9.0%, 31.7% to 36.7%, 36.1% to 40.4%, and 29.6% to 33.8% in comparison with the other three algorithms in random, tree, reverse tree and diamond structured workflows, respectively. Here, BAWMEE fails to save energy in chain-structured workflows, because the deadline baseline is set too tight for this structure based on our deadline generation method, as indicated by the average number of tasks per job and the average workload reduction of BAWMEE in Figure 5.29. BAWMEE almost misses no deadlines except in tree-structured workflows, where it favors the jobs close to the root more than those close to leaves and thus leads to an unfair division of the slack time [83]. Besides, the URT of BAWMEE is less than 0.8 seconds, and is 31.2% to 5.4, 1.2 to 4.1, and 87.4% to 2.3 times of those of EEDAW, MinD+ED, and MinD+EEDAJ with different workflow structures, respectively.

5.6 Conclusion

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

CHAPTER 6

ENERGY-EFFICIENT DYNAMIC SCHEDULING OF MAPREDUCE WORKFLOWS IN SHARED CLUSTERS

This chapter is organized as follows. Section 6.1 introduces the significant impact of dynamic scheduling of MapReduce workflows on energy efficiency. Section 6.2 formulates a dynamic big data workflow scheduling problem. Section 6.3 discusses the algorithm design principles of the scheduling problem. We design a heuristic for the problem and the corresponding system modules for algorithm implementation in Section 6.4. Section 6.5 presents performance evaluation. Section 6.6 concludes our work.

6.1 Introduction

Big data analytics require the invocation and coordination of a large collection of 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 in data centers.

In Section 3.2, we validate with experimental results that the DEC of a MapReduce job 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 two commonly used green computing techniques, i.e., task consolidation to reduce SEC by turning off idle servers and load balancing to reduce DEC through DVFS,

and would add an additional level of energy efficiency to current computing platforms processing big data workflows.

Task scheduling algorithms are divided into two categories: static and dynamic scheduling algorithms. The former determines task mapping onto resources before the execution of the entire application, based on accurate information about task execution cost, which is supposed to be known at compilation time. The latter schedules tasks to resources in the runtime to flexibly optimize certain goals on line, and have much lower or no requirements on the accuracy of *a priori* knowledge, so that it is more widely applied to practical systems. In Chapter 5, 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 chapter, 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 our approach in Chapter 5 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 the schedulers in [22] and [23], 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 for static mapping of each workflow on a virtual homogeneous cluster to determine the task partitioning of each component job, ii) Phase II for static mapping

of each workflow on an idle heterogeneous cluster to set the virtual deadline of each component job, and iii) Phase III for 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 in a workflow engine, such as Oozie and Tez, from the perspective of the entire workflow; the last dynamic subproblem is naturally 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 consider a set of MapReduce workflows and propose a semi-dynamic online scheduling algorithm, which adaptively reduces DEC from a global perspective in both temporal and spatial aspects, and explicitly accounts for execution time estimation inaccuracies and computing system dynamics.
- The performance superiority of the proposed algorithm in terms of dynamic energy saving and deadline missing rate is illustrated by experimental results using the Oozie workflow engine in Hadoop/YARN systems and extensive simulation results in comparison with existing algorithms.

6.2 Problem Formulation

6.2.1 Cost Models

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.

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 DAG $G_j(V_j, A_j)$, where each vertex $v_{j,k} \in V_j$ represents a component job, and each directed edge $a_{j,k,k'} \in A_j$ denotes an execution dependency. We consider the map and reduce phases of each MapReduce 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.

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 [61, 66], 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 $n_i(t) = \sum_{f_j \in F} \sum_{v_{j,k} \in f_j} n_{i,j,k}(t)$ 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.

Energy Model The DEC of a cluster is

$$E = \sum_{m_i \in M} \int_0^T P_i \sum_{f_j \in F} \sum_{v_{j,k} \in f_j} [\mu_{i,j,k} n_{i,j,k}(t)] dt,$$

Table 6.1 Notations Used in the Cost Models

Notations	Definitions
M	a cluster of machines
$m_i(N_i, p_i, o_i, P_i)$	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
$f_j(G_j(V_j, A_j), t_j, d_j)$	the j-th workflow request consisting of a workflow structure of a DAG $G_j(V_j, A_j)$ and a deadline d_j arrives at time t_j
t_j^C	the completion time of workflow f_j
$v_{j,k}$	the k-th component job in workflow f_j
$a_{j,k,k'}$	the directed edge from job $v_{j,k}$ to job $v_{j,k'}$
$t_{j,k}^S$, $t_{j,k}^F$	the actual start and finish time of job $v_{j,k}$
$\mu_{i,j,k}$	the percentage of time executing CPU-burst instructions in job $v_{j,k}$ on machine m_i
$o_{j,k}$	the memory demand per task in job $v_{j,k}$
$s_{j,k,l}$	the l-th task in job $v_{j,k}$
$L_{j,k}$, $L'_{j,k}$	the number and the maximum possible number of tasks in $v_{j,k}$
$w_{j,k}(L_{j,k})$	the workload of job $v_{j,k}$ partitioned into $L_{j,k}$ tasks
$w_{j,k,l}(L_{j,k})$	the workload of task $s_{j,k,l}$ in $v_{j,k}$ with $L_{j,k}$ tasks
$t_{i,j,k,l}$	the execution time of task $s_{j,k,l}$ running on machine m_i
$n_{i,j,k}(t)$	the number of tasks in job $v_{j,k}$ running on machine m_i at time t
$n_i(t)$	the number of used CPU cores on machine m_i at time t
$o_i(t)$	the size of used memory on machine m_i at time t
T	a time period of the cluster's operation
E	the dynamic energy consumption of cluster M

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 [22].

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.

6.2.2 Problem Definition

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

Definition 8. ***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 6.2 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

6.3 The Design Principles of the Scheduler

We first summarize the design of four algorithms adapted from different scenarios: i) BAWMEE in Chapter 5 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 [22] and [23]) 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 [83] 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 6.4.2 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 6.2 between the aforementioned four algorithms and our proposed method in Section 6.4.

6.3.1 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 Chapter 5 repeatedly performs static mapping of each workflow. We adopt dynamic scheduling to consider resource sharing among multiple workflows from a global perspective.

6.3.2 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.

6.3.3 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 [83]. 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 execute 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.

6.4 Algorithm and System Design

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

6.4.1 DAWSEE Overview

The workflow scheduling process consists of three components as shown in Figure 6.1: 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 [83], 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 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 for energy saving. Once a job is ready to execute, DEETS allows its tasks

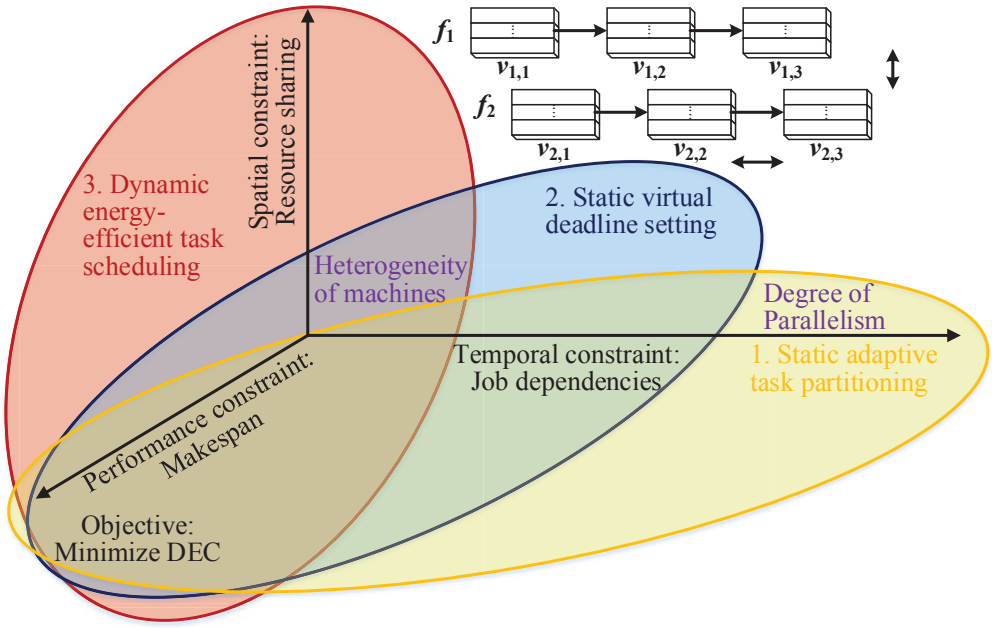


Figure 6.1 Decoupling and semi-dynamic scheduling.

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.

A data center typically has local storage attached to computing nodes, which are connected via high-speed switches. The workflow engine, such as Oozie [3], on Hadoop/YARN is responsible for handling MapReduce workflow requests. As shown in Figure 6.2, 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

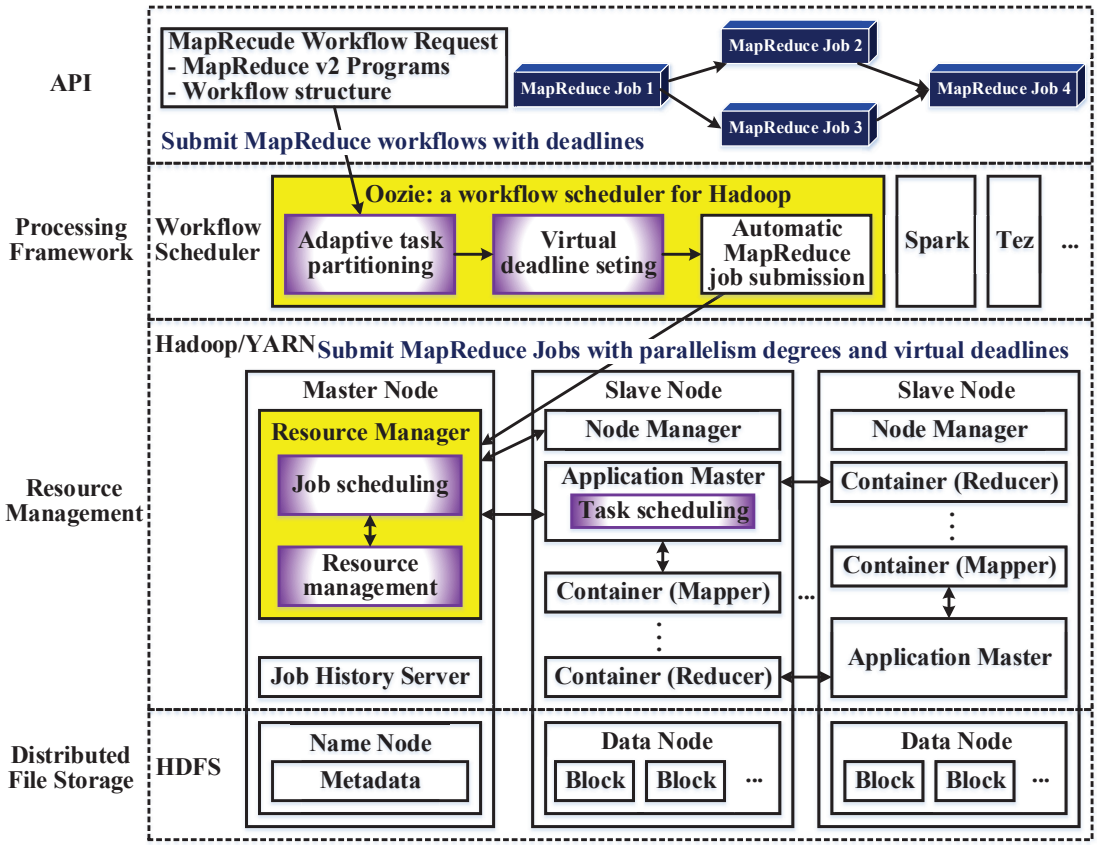


Figure 6.2 The architecture of the MapReduce workflow scheduling system.

module sits. This architecture keeps the workflow engine and the Hadoop system loosely coupled and is best suited for our algorithm implementation.

6.4.2 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 Algorithm 17). Here, the 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, i.e., determining the number of tasks in each job but without the actual mapping of jobs to a specific machine at the time of workflow submission, and compute its virtual start time (VST) $t_{j,k}^{VS}$ and virtual finish time (VFT) $t_{j,k}^{VF}$. If all the preceding jobs of job $v_{j,k}$ are virtually mapped, its earliest virtual start time (EVST) $t_{j,k}^{ES}$ 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) $t_{j,k}^{LF}$ 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.

Algorithm 17 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 Algorithm 18. 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, Algorithm 17 terminates after at most $|V_j|$ iterations.

Algorithm 17: 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 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$\}$;

In Algorithm 18 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 $\{(\tilde{t}_{j,k}(L), \tilde{e}_{j,k}(L))|L \in [1, L'_{j,k}]\}$, and use an FPTAS based on Algorithm 10 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 Line 12). If there exists a job that violates the precedence constraints, we divide the pipeline at this job, and recursively call Algorithm 18 to compute the virtual mapping of the upstream sub-pipeline with updated EVST and LVFT constraints (in Lines 13-20).

Algorithm 18: 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: $t^{VS}_{j,1} \leftarrow pl.evst$; $t^{VF}_{j,1} \leftarrow t^{VS}_{j,1} + \tilde{t}_1(L'_{j,1})$; $L_{j,1} \leftarrow L'_{j,1}$;

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 Algorithm 10 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.

6.4.3 Virtual Deadline Setting

Initially, all jobs are supposed to run the fastest on their respective machines. 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 deadline extension. Then, the rest of the jobs are considered in the 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. In MinD, please refer to [83] about applying equal slack time to deadlines in the following step.

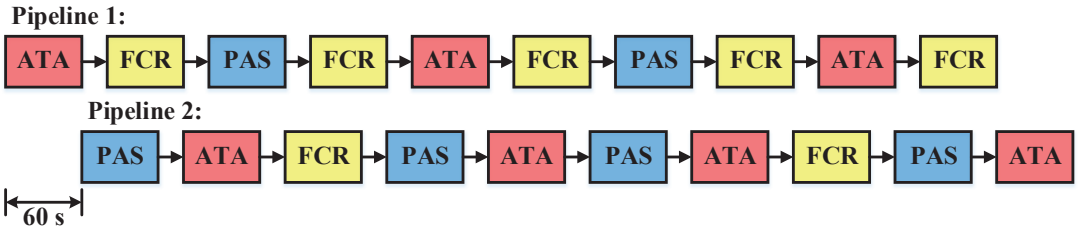


Figure 6.3 Pipeline-structured MapReduce workflows.

6.5 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.

6.5.1 Experiments

Experimental Settings The testbed is the same as described in Subsection 3.2.1. On the cluster in our testbed, we also install Oozie 4.3 [3], 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 Figure 5.5. 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

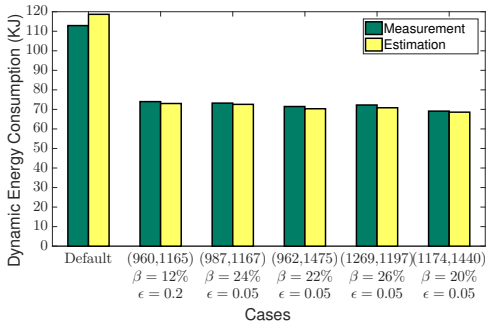


Figure 6.4 The DEC of a pair of pipelines with different approximate ratios under different deadline constraints.

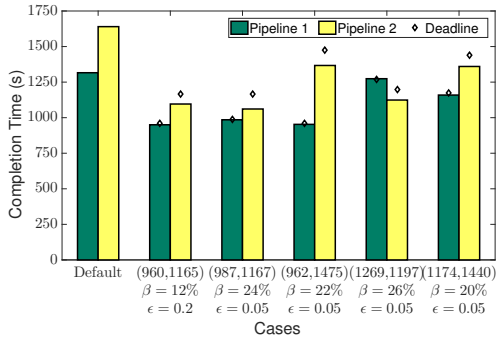


Figure 6.5 The completion time of a pair of pipelines with different approximate ratios under different deadline constraints.

default settings in Hadoop, where the number of mappers is the input size divided by the split size of 128 MB, and the number of reducers is 1. Hence, we refer to the mapping scheme produced by these two algorithms as the "default" scheme in this scenario.

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 Figure 6.3, and plot in Figures 6.4 and 6.5 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.

6.5.2 Simulation

Simulation Settings We generate a set of random workflows using the method in Subsection 5.5.2. 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} flops when running in serial. Based on our measurements in Subsection 3.2.2, the workload $w(k)$ 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 critical path 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 5.5, based on four 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 scheduling simulation lasts for 3 days and is repeated 20 times with different workflow instances, whose arrivals follow Poisson distribution. In the performance 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. 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.

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 Figures 6.6-6.9, respectively. We observe that the DMR of all the algorithms

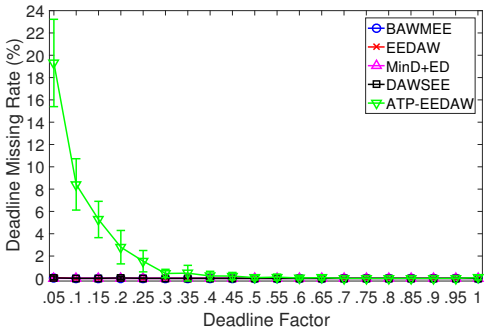


Figure 6.6 The DMR vs. deadlines.

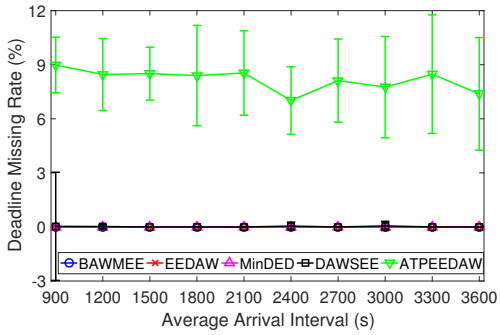


Figure 6.7 The DMR vs. arrival intervals.

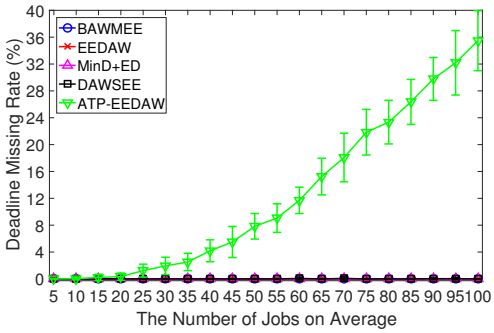


Figure 6.8 The DMR vs. workflow sizes.

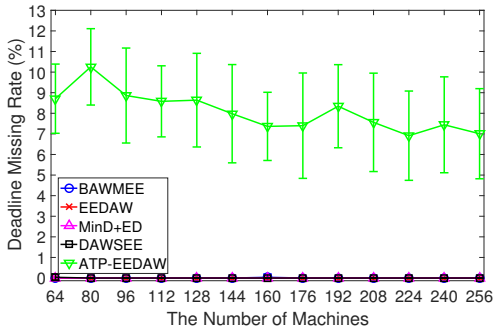


Figure 6.9 The DMR vs. cluster sizes.

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.

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 Figure 6.10, which shows that DAWSEE saves DEC

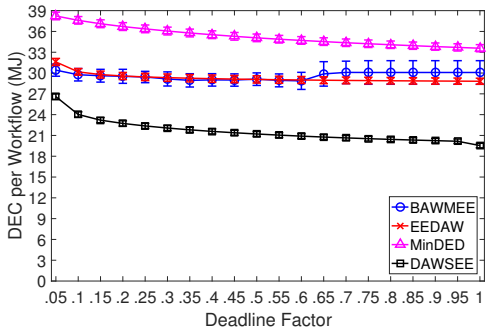


Figure 6.10 The DEC vs. deadlines.

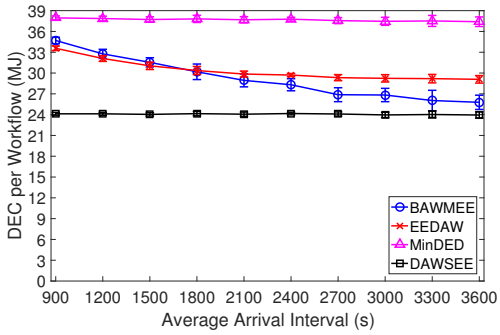


Figure 6.11 The DEC vs. arrival intervals.

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.

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 Figure 6.11, 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 Figure 6.12, 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

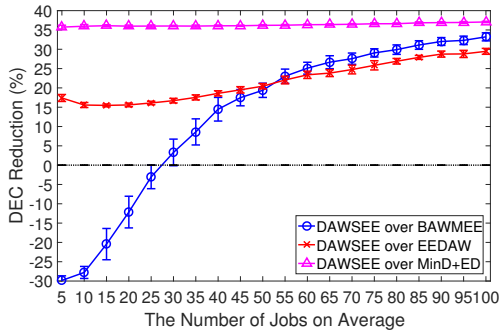


Figure 6.12 The DECR vs. workflow sizes.

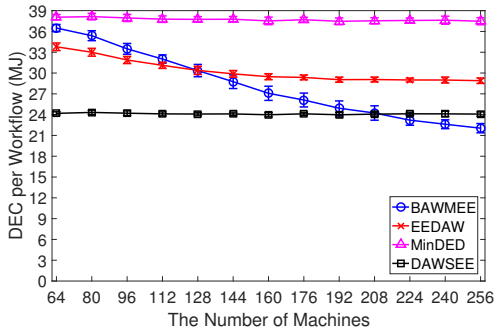


Figure 6.13 The DEC vs. cluster sizes.

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.

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 Figure 6.13, 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 increases to 224 machines and beyond, with sufficient resources, the energy saving from balancing the resource use among workflows in DAWSEE is less than that from joint optimization of task partitioning and resource allocation within a single workflow in BAWMEE. the superior performance of BAWMEE over DAWSEE for large cluster sizes is not caused by the difference in the number of tasks.

6.6 Conclusion

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.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

7.1.1 Achievements

Performance Optimization of MapReduce Workflow Mapping in Clouds Cloud computing provides a cost-effective computing platform for big data workflows where moldable parallel computing models such as MapReduce are widely applied to meet stringent performance requirements. The granularity of task partitioning in each moldable job has a significant impact on workflow completion time and financial cost that is decided by the total workload. We designed a big-data workflow mapping model, based on which, we formulated a strongly NP-complete problem of workflow mapping to minimize workflow makespan under a budget constraint in public clouds. We designed an FPTAS for a special case with a pipeline-structured workflow executed on virtual machines of a single class and a heuristic for the generalized problem with an arbitrary DAG-structured workflow executed on virtual machines of multiple classes.

Energy Efficiency of MapReduce Workflow Mapping/Scheduling in Shared Clusters Large-scale workflows for big data analytics have become a main consumer of energy in data centers. Such big data workflows are typically comprised of MapReduce programs, which are moldable parallel jobs running on any number of processors decided prior to execution. However, most of the existing efforts on energy-efficient computing were focused on independent MapReduce jobs and workflows comprised of serial jobs. The widely adopted energy-saving techniques, including task consolidation to reduce SEC by turning off idle servers and load balancing to reduce DEC through DVFS, are not sufficient to address the energy efficiency issue of big data workflows. Therefore, we directed our efforts to workflow

scheduling for dynamic energy saving by adaptively determining the degree of parallelism in each job to mitigate the workload overhead while meeting a given performance requirement. A moldable job typically follows a performance model where the workload of each component task decreases and the total workload, which decides DEC, increases as the number of parallel component tasks increases. This model was validated with experimental results and served as a base of our workflow scheduling solutions for energy saving of big data workflows.

We formulated a workflow scheduling problem to minimize the DEC of a given workflow request under deadline and resource constraints in a Hadoop cluster, which has been shown to be strongly NP-hard. We started with a special case with a chain of moldable jobs on a homogeneous cluster, which has been proved to be weakly NP-complete and solved by an FPTAS of linear time complexity with respect to $1/\epsilon$. By leveraging the near optimality and low time complexity of this FPTAS, we designed a heuristic for the generalized problem of statically scheduling a DAG-structured workflow on a heterogeneous cluster. Static scheduling typically requires *a priori* knowledge of exact job execution cost and an accurate snapshot of available computing resources to schedule an individual MapReduce job, while dynamic scheduling may lack a global perspective to balance the trade off between energy cost and execution time of component jobs in each workflow. Therefore, we also proposed a semi-dynamic online scheduling algorithm, which adaptively reduces the DEC of a set of MapReduce workflows while meeting performance requirements from a global perspective, and explicitly accounts for inaccurate execution time estimates and computing system dynamics. The performance superiority of the proposed solutions was illustrated by extensive simulation and experimental results in comparison with existing algorithms.

7.1.2 Discussion

Novelty In this dissertation, we optimize the mapping feasibility, workflow makespan, and energy efficiency of big-data scientific workflows by adaptively determining the degree of parallelism in each MapReduce job to reduce workload overhead. Our approach is orthogonal to the existing techniques, and would add an additional level of intelligence to the current computing platforms executing big data workflows.

Contributions The MapReduce workflow mapping problem for makespan minimization in clouds and the MapReduce workflow scheduling problem for energy efficiency in shared clusters are both strongly NP-complete. We proved that their special case of a pipeline-structured workflow on homogeneous machines is weakly NP-complete, and solved it by an FTPAS of linear time complexity with respect to $1/\epsilon$. Based on the insights into the computational complexity of the aforementioned problems, we leveraged the properties of near optimality and low time complexity of the FPTAS in the design of heuristics for the generalized problems, yielding superior performances over existing algorithms.

Application Scope Our approach of adjusting the degree of parallelism in each MapReduce job in a workflow does not require any support from extra hardware and third-party service providers, and thus can be applied to different computing environments, such as public clouds and shared clusters. In addition to the performance measurements on several standard big data benchmarks, such as Word-Count and Grep in Section 3.2, we carefully investigated other big data benchmarks, such as BigBench and TeraSort. The measurements of execution time and energy consumption with different degrees of parallelism show that BigBench and TeraSort do not follow the performance model of a moldable job. We observed that most of the instructions in BigBench are actually I/O-bound. Hence, BigBench that reads data from a single disk may not be considered as a typical parallel computing

program because I/O operations on the same disk are not performed in parallel. Also, we observed that the map and reduce functions of TeraSort are empty, and the majority of the workload in TeraSort occurs in the shuffling phase. We would like to point out that our approach is most suitable for big data workflows comprised of CPU-bound computing jobs whose main workloads are in the map and reduce phases.

In the Hadoop/Spark system, the reduction of job workload with a decreasing number of parallel component tasks is attributed to less communication between the resource manager and node managers for scheduling tasks and less overhead for launching containers to execute tasks. In our experimental testbed, the highest degree of parallelism supported by the system is only 23 and is considered small in comparison with large-scale production computing systems. For a MapReduce job with no more than 23 parallel tasks to process a large dataset, the task execution time is far more than the overhead, so the workload variation is not very obvious with different degrees of parallelism. However, in large-scale production systems, we believe that adaptively determining the number of parallel component tasks in a MapReduce job would significantly reduce the total workload. In addition, the degree of parallelism should have a lower bound to limit the longest execution of a task because resuming a failed task would involve a significant amount of workload.

Observations In our experimental results, we observed that the number of killed tasks slightly varies in different runs of the same MapReduce job on the same input data, and is closely related to the preset number of parallel tasks. This is an interesting phenomenon and deserves further efforts for exploration.

7.2 Future Work

With the ever-increasing data volume, velocity, variety, and veracity, domain experts in various scientific fields will be facing more challenges along the line of our research in the coming years as follows.

In addition to CPU-bound MapReduce applications, there exist many I/O-bound MapReduce applications, such as BigBench, which cannot be optimized based on our approach in this dissertation. Therefore, it is critical to model and optimize the performance and energy efficiency of I/O-bound MapReduce applications and workflows comprised of such type of applications.

Spark on Hadoop/YARN based on the MapReduce framework is gaining its popularity because it provides convenient programming APIs and supports the processing of interactive and streaming data. Meanwhile, high-performance computing paradigms such as MPI, OpenMP, CUDA, and OpenCL have flexible computing and message passing models for accelerating complex computing. When big data meet high-performance computing, performance optimization that considers the integration of distributed data storage and parallel computing becomes a more complex problem.

In machine learning and data mining, various intelligent classifiers have been widely applied to next-generation scientific applications and their deep learning strategies heavily rely on large-scale neural networks, which could be modeled as big data workflows with layered structures. It is of paramount importance to optimize the performance of and reduce the energy consumption of such workflows for computational intelligence by exploiting integrated parallel computing and big data technologies.

BIBLIOGRAPHY

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

[2] 2016. Apache Hadoop. http://hadoop.apache.org.

[3] 2016. Apache Oozie. https://oozie.apache.org.

[4] 2017. Atmospheric Sciences Division, Brookhaven National Laboratory. http://www.ecd.bnl.gov/aboutASD.html.

[5] 2017. Climate Change Science Institute, Oak Ridge National Laboratory. https://climatechangescience.ornl.gov.

[6] 2017. Spallation Neutron Source. https://neutrons.ornl.gov/sns.

[7] 2017. Relativistic Heavy Ion Collider, Brookhaven National Laboratory. http://www.bnl.gov/rhic.

[8] 2017. PUMA Benchmarks and dataset downloads. https://engineering.purdue.edu / puma/datasets.htm.

[9] S. Abrishami and M. Naghibzadeh. Deadline-constrained workflow scheduling in software as a service cloud. *Scientia Iranica*, (0), 2012.

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

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

[12] 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.

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

[14] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report No. UCB/EECS-2009-28, Reliable Adaptive Distributed Systems Laboratory, University of California, Berkeley, Feb 2009.

[15] R. Bajaj and D. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE TPDS*, 15(2):107–118, Feb. 2004.

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

[17] 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.

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

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

[20] 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.

[21] Y. Chen, L. Keys, and R.H. Katz. Towards energy efficient MapReduce. Technical Report No. UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.

[22] 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.

[23] 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.

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

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

[26] J.J. Durillo, V. Nae, and R. Prodan. Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff. In *Proc. of IEEE/ACM CCGrid*, pages 203–210, Delft, Netherlands, May 2013.

[27] J.J. Durillo, V. Nae, and R. Prodan. Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Elsevier FGCS*, 36:221–236, 2014.

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

[29] B. Flaugher. The dark energy survey camera (decam). *Bulletin of the American Astronomical Society*, 42(406), 2010.

[30] 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.

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

[32] Y. Gu and Q. Wu. Optimizing distributed computing workflows in heterogeneous network environments. In *Proc. of the 11th Int. Conf. on Distributed Computing and Networking*, Kolkata, India, Jan. 3-6 2010.

[33] Y. Gu, Q. Wu, and N.S.V. Rao. Analyzing execution dynamics of scientific workflows for latency minimization in resource sharing environments. In *Proc. of the 7th IEEE World Congress on Services*, Washington DC, Jul. 4-9 2011.

[34] T. Hacker and K. Mahadik. Flexible resource allocation for reliable virtual cluster computing systems. In *Proc. of the ACM/IEEE Supercomputing Conference*, pages 48:1–48:12, 2011.

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

[36] R. Huang and H. Casanova. Automatic resource specification generation for resource selection. *Proceedings of the 2007 ACM/IEEE*, 2007.

[37] 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.

[38] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *Proc. of SPAA*, pages 86–95, Las Vegas, Nevada, USA, Jul 2005.

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

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

[41] 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.

[42] K.R. Krish, M.S. Iqbal, M.M. Rafique, and A.R. Butt. Towards energy awareness in Hadoop. In *Proc. of Int. Work. on Network-Aware Data Management*, pages 16–22, New Orleans, LA, USA, Jun-Jul 2014.

[43] 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.

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

[45] 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.

[46] 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.

[47] Y.C. Lee and A.Y. Zomaya. Stretch out and compact: Workflow scheduling with resource abundance. In *Proc. of IEEE/ACM CCGrid*, pages 203–210, Delft, Netherlands, May 2013.

[48] 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.

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

[50] P. Li, L. Ju, Z. Jia, and Z. Sun. SLA-aware energy-efficient scheduling scheme for Hadoop YARN. In *Proc. of IEEE HPCC*, pages 623–628, New York, USA, Aug 2015.

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

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

[53] M. Mao, J. Li, and H. Marty. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing*, pages 41–48, Oct. 2010.

[54] Y. Mao, W. Wu, H. Zhang, and L. Luo. GreenPipe: a Hadoop based workflow system on energy-efficient clouds. In *Proc. of IPDPS Work. and PhD Forum*, pages 2211–2219, Shanghai, China, May 2012.

[55] 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.

[56] 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.

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

[58] M. Rahman, S. Venugopal, and R. Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *IEEE Int. Conf. on e-Science and Grid Comp.*, pages 35–42, Dec. 2007.

[59] A. Rasmussen, K. Gilmore, S.M. Kahn, J. Geary, S. Marshall, M. Nordby, P. O'Connor, S. Olivier, J. Oliver, V. Radeka, T. Schalk, R. Schindler, J. Tyson, R.V. Berg, and LSST Camera Team. The camera for LSST and its focal plane array. *Bulletin of the American Astronomical Society*, 41(221), 2010.

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

[61] F. Arickx 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.

[62] R. Sakellariou and H. Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Proc. of the 18th IEEE IPDPS*, volume 2, page 111, 2004.

[63] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos. Scheduling workflows with budget constraints. In Sergei Gorlatch and Marco Danelutto, editors, *Integrated Research in GRID Computing*, pages 189–202. Springer US, 2007.

[64] 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.

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

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

[67] B. Sharma, T. Wood, and C.R. Das. HybridMR: A hierarchical mapreduce scheduler for hybrid data centers. In *Proc. of IEEE ICDCS*, pages 102–111, Philadelphia, PA, USA, Jul 2013.

[68] H. Topcuoglu, S. Hariri, and M.Y. Wu. Performance effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS*, 13(3):260–274, 2002.

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

[70] Y. Wang and W. Shi. On scheduling algorithms for MapReduce jobs in heterogeneous clouds with budget constraints proceeding. In *Proc. of OPODIS*, pages 251–265, Nice, France, Dec 2013.

[71] Y. Wang and W. Shi. Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds. *IEEE Tran. on Cloud Comp.*, 2(3):306–319, 2014.

[72] C.Q. Wu and H. Cao. Optimizing the performance of big data workflows in multi-cloud environments under budget constraint. In *Proc. of IEEE SCC*, pages 138–145, San Francisco, CA, USA, Jun - Jul 2016.

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

[74] Q. Wu and Y. Gu. Optimizing end-to-end performance of data-intensive computing pipelines in heterogeneous network environments. *J. of Parallel and Distributed Computing*, 71(2):254–265, 2011.

[75] 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.

[76] J. Yu. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Proc. of SC WORKS Workshop*, pages 1–10, 2006.

[77] J. Yu, R. Buyya, and C. Tham. Cost-based scheduling of scientific workflow application on utility grids. In *Proc. of the 1st Int. Conf. on e-Science and Grid Comp.*, pages 140–147, Washington, DC, USA, 2005.

[78] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, 26(8):1200–1214, October 2010.

[79] 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.

[80] 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.

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

[82] 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.

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