Multi-Objective Scheduling Algorithm for DAG Tasks in Cloud Computing Environments: Post-print

Authors: Xu Jianrui, Zhu Huijuan

Date: 2018-05-20T00:00:00+00:00

Abstract

To achieve simultaneous optimization of task execution efficiency and execution cost, a multi-objective scheduling optimization algorithm for DAG tasks in cloud computing environments is proposed. The algorithm models the multi-objective optimization problem as a set of balanced optimal solutions satisfying Pareto optimality and solves the model heuristically; simultaneously, to evaluate the quality of multi-objective balanced solutions, an evaluation mechanism based on the hypervolume method is designed, thereby obtaining balanced scheduling solutions among conflicting objectives. Through simulation experimental tests configured with cloud environments and three synthetic workflows and two real-world scientific workflows, the results demonstrate that, compared with similar single-objective algorithms and multi-objective heuristic algorithms, the algorithm not only achieves higher solution quality but also exhibits better solution balance, better aligning with the resource usage characteristics and workflow scheduling patterns of real-world clouds.

Full Text

Preamble

Multi-objective Scheduling Algorithm for DAG Tasks in Cloud Computing Environments

Xu Jianrui[1],[2], Zhu Huijuan[3]

[1]School of Computer Science & Telecommunication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212013, China;
[2]Zhenjiang Branch, Jiangsu Union Technical Institute, Zhenjiang, Jiangsu 212016, China;

[3]School of Computer & Control Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract: To achieve simultaneous optimization of task execution efficiency and execution cost, this paper proposes a multi-objective scheduling optimization algorithm for DAG tasks in cloud computing environments. The algorithm models the multi-objective optimization problem as a set of trade-off optimal solutions satisfying Pareto optimality and solves the model using a heuristic approach. Simultaneously, to evaluate the quality of multi-objective trade-off solutions, we design an assessment mechanism based on the hypervolume method, thereby obtaining balanced scheduling solutions among conflicting objectives. Through simulation experiments configured with a cloud environment, three types of synthetic workflows, and two real-world scientific workflows, the results demonstrate that compared with similar single-objective algorithms and multi-objective heuristic algorithms, our algorithm not only achieves higher solution quality but also provides better solution balance, making it more aligned with real-world cloud resource usage characteristics and workflow scheduling requirements.

Keywords: cloud computing; workflow scheduling; multi-objective optimization; Pareto front; Amazon EC2

---

0 Introduction

Scientific experiments in various domains are typically defined as workflows, where tasks form chain-like structures based on data flow and computational dependencies, such as compute-intensive and data-intensive workflow applications with massive data volumes and computational requirements that demand high-performance computing environments for execution [1]. The emergence of cloud computing provides more efficient technical support and execution environments for scientific workflows [2]. Cloud offers computing resources in the form of virtual machines (VMs), and the mapping problem between tasks in workflows and computing resources constitutes the workflow scheduling problem. Cloud workflow scheduling involves two levels: first, mapping between tasks and VMs, and second, sequential execution of tasks on individual VMs [3]. This paper addresses the multi-objective optimization problem of cloud workflow scheduling based on the Amazon EC2 environment, focusing on optimizing both workflow execution makespan and execution cost to find optimal scheduling solutions.

In related research, reference [4] proposes a Heterogeneous Budget-Constrained Scheduling (HBCS) algorithm that adjusts the ratio between available budget and minimum price through a cost factor to achieve optimized scheduling. Reference [5] presents a Heterogeneous Earliest Finish Time (HEFT) algorithm that minimizes task scheduling time by assigning different priorities to tasks. Reference [6] introduces a Budget-constrained HEFT (BHEFT) algorithm, which

extends HEFT to consider optimal budget constraints in task scheduling. Reference [7] proposes a Cloud Workflow Discrete Particle Swarm Optimization (CWDPSO) algorithm that uses particle swarm evolution to solve resource allocation schemes with resource usage cost and security satisfaction as optimization objectives. All four algorithms—HEFT, BHEFT, HBCS, and CWDPSO—convert the optimization objective into single-objective optimization and are not suitable for Amazon EC2 cloud workflow scheduling scenarios.

In multi-objective scheduling, reference [8] proposes the POSH algorithm, which simultaneously considers execution makespan and cost while using user-defined preference factors to set weights for both objectives, achieving multi-objective optimization. The NSPSO algorithm in reference [9] and the Fussy-PSO algorithm in reference [10] both attempt to use particle swarm optimization algorithms to achieve a balance between workflow scheduling makespan and execution cost. Reference [11] proposes a Multi-objective HEFT (MOHEFT) algorithm, which aims to extend HEFT for multi-objective optimization through Pareto-linear heuristic algorithms. However, the aforementioned multi-objective scheduling algorithms are only applicable to traditional heterogeneous computing environments, as IaaS cloud environments differ from traditional computing environments in terms of computing models, data, and pricing models.

This paper proposes a Cloud Multi-objective HEFT (CMOHEFT) algorithm for workflow scheduling in cloud environments. The algorithm targets the simultaneous minimization of workflow task execution makespan and economic cost, obtains a set of balanced optimal solutions among multiple objectives through heuristic methods, and evaluates the effectiveness of these balanced solutions using Pareto fronts and hypervolume values.

---

1 Amazon EC2 Cloud Environment

Amazon Elastic Compute Cloud (EC2) is an Infrastructure-as-a-Service (IaaS) cloud service that openly provides users with Amazon' s computing facilities. The "elastic" characteristic indicates that users can expand and contract their infrastructure by requesting or releasing resources and instances. Currently, Amazon EC2 offers three types of instances: (a) Reserved Instances, which allow users to make long-term reservations for multiple host resources; (b) On-Demand Instances, which allow users to pay only for requested resources and time, with billing units in hours; and (c) Spot Instances, which allow users to bid on unused Amazon resources by specifying a maximum bid price. If the bid exceeds the current spot price (which depends on resource demand), the user obtains the instance; if the bid falls below the spot price, Amazon reclaims the instance.

Amazon EC2 provides 14 different types of on-demand instances with varying performance and costs. Instance computing performance is defined according to Elastic Compute Units (ECUs), equivalent to the capacity of a Xeon pro-

cessor with 1.0-1.2 GHz. Table 1 summarizes the average performance of these resources, the cost per hour of computing capacity, and the number of floating-point operations per second per unit cost (millions of floating-point operations per second). The multi-objective workflow scheduling algorithm in this paper will evaluate performance using the five instance types shown in Table 1.

---

2 Model Description

2.1 Workflow DAG Task Model

A workflow application is defined as a Directed Acyclic Graph (DAG), $W = (A, D)$, where A represents a set of n tasks, $A = \cup_{i=1}^{n}\{A_i\}$, and D represents control-flow and data-flow dependencies between tasks, $D = \{(A_i, A_j, Data_{ij})|(A_i, A_j) \in A \times A\}$. $Data_{ij}$ denotes the data transfer volume between tasks A_i and A_j. Let $pred(A_i) = \{A_k|(A_k, A_i, Data_{ki}) \in D\}$ represent the predecessor task set of task A_i, which must complete before A_i can begin. Assume the computational load of each task A_i is known and expressed as the number of executable machine instructions.

2.2 Resource Model

Assume the cloud platform consists of a set of m heterogeneous resources, $R = \cup_{j=1}^{m}\{R_j\}$, provided by Amazon EC2 cloud resource types such as m1.small, m1.medium, m1.large, m1.xlarge, c1.medium, and c1.xlarge. For a given resource R_j of a determined type, its average performance is measured in GFLOPs (Floating-point Operations Per Second). In this paper' s workflow model, we assume workflow tasks can execute in parallel on Amazon EC2 VM instances as shown in Table 1 , with resource usage charged per hour (third column of Table 1). The final user cost depends not only on resource usage but also on data storage and data transfer between different VM instances, specifically including four components: per-hour resource usage price PER_i; per-MB data storage price PSR_i; per-MB data input price PIR_i; and per-MB data output price POR_i.

2.3 Problem Definition

The workflow scheduling problem in this paper involves scheduling workflow tasks to execute on Amazon cloud resources to minimize both execution makespan and economic cost. Let $sched(A_i)$ denote the execution resource for task A_i. We define two optimization objectives as follows:

1) Execution Time Makespan

Let *execution time* $t(A_i, R_j)$ represent the sum of computation time for task A_i on resource $R_j = sched(A_i)$ and the maximum input data reception time from any predecessor task $A_p \in pred(A_i)$, expressed as:

$$t(A_i, R_j) = \frac{workload(A_i)}{s_j} + \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{pj}} \right\}$$

where $Data_{pi}$ denotes the data transfer volume between tasks A_p and A_i, b_{pj} represents the bandwidth between the execution resource of task A_p and resource R_j, $workload(A_i)$ denotes the instruction length of task A_i, and s_j denotes the computing speed of resource R_j in MIPS (Million Instructions Per Second). The completion time TA_i of task A_i includes both the task' s own execution time and its predecessors' execution times, expressed as:

$$TA_i = \begin{cases} t(A_i, sched(A_i)) & \text{if } pred(A_i) = \emptyset \\ \max_{A_p \in pred(A_i)} \{TA_p + t(A_i, sched(A_i))\} & \text{if } pred(A_i) \neq \emptyset \end{cases}$$

The workflow execution makespan is defined as the maximum completion time among all tasks:

$$makespan = \max_{i \in [1,n]} TA_i$$

2) Economic Cost

The economic cost of workflow execution consists of two parts: computation cost $C(comp)$ and data transfer/storage cost $C(data)$. Let $C(data)(A_i, R_j)$ denote the cost of transferring input data $In(A_i)$, output data $Out(A_i)$, and storing data $Data(A_i)$ during task A_i execution on resource R_j:

$$C(data)(A_i, R_j) = PIR_i \times In(A_i) + POR_i \times Out(A_i) + PSR_i \times Data(A_i)$$

Let $C(comp)_{R_j}$ denote the cost of using resource R_j. For resource R_j executing task A_i, let $t(start)_{A_i}$ denote the task' s start time and $t(end)_{A_i}$ denote its end time. Assume the time for transferring input data $In(A_i)$ and output data $Out(A_i)$ occurs between $t(start)_{A_i}$ and $t(end)_{A_i}$.

Consider a set of p tasks $\{J_1, J_2, ..., J_p\}$ scheduled on resource R_j, where $p < n$ and $sched(J_i) = R_j$ for $i \in [1, p]$. Tasks are ordered by start time: $t(start)_{J_1} < t(start)_{J_2} < ... < t(start)_{J_p}$. Based on this ordering, all tasks are divided into q ($q \leq p$) different groups $G_k^{(j)}$, $1 \leq k \leq q$, such that all tasks within the same group can execute consecutively without releasing the resource. The resource can be released after the task with the maximum start time in a group completes.

The first group $G_1^{(j)} = \{J_1, J_2, ..., J_r\}$ is constructed through three rules:

Rule 1: The first task J_1 belongs to the first group: $J_1 \in G_1^{(j)}$.

Rule 2: Each task $J_i \in G_1^{(j)}$ $(2 \leq i \leq r)$ completes before resource release. This indicates that J_i can start while the resource is still rented due to J_{i-1}' s execution:

$$t(start)_{J_i} < t(end)_{J_{i-1}} + 3600$$

Rule 3: The next task $J_{r+1} \notin G_1^{(j)}$ $(r+1 \leq p)$ begins execution immediately at start time $t(start)_{J_{r+1}}$ when the resource has been released, meaning that after task J_r completes, the final one-hour rental period for executing J_r has expired, and resource R_j is idle between $t(end)_{J_r}$ and $t(start)_{J_{r+1}}$:

$$\left\lfloor \frac{t(end)_{J_r}}{3600} \right\rfloor \times 3600 < t(start)_{J_{r+1}}$$

Continuous groups are established until the final task J_p is assigned to a group. The second group $G_2^{(j)}$ is constructed similarly using J_{r+1} instead of J_1, and the same strategy is applied to build remaining groups. After establishing all groups, the cost $C(comp)_{R_j}$ for using resource R_j is defined as the product of the number of hours required to execute all groups and the per-hour cost:

$$C(comp)_{R_j} = \sum_{k=1}^{q} \left\lceil \frac{\sum_{J_i \in G_k^{(j)}} t(A_i, R_j)}{3600} \right\rceil \times PER_j$$

The economic cost for executing workflow $W = (A, D)$ is the sum of computation costs and data transfer/storage costs across all m resources:

$$Cost(W, R) = \sum_{j=1}^{m} C(comp)_{R_j} + \sum_{(A_i, A_j, Data_{ij}) \in D} C(data)(A_i, R_j)$$

3 Multi-objective Optimization

Multi-objective optimization problems are typically defined as finding all solutions x that minimize a vector function $f(x) = [f_1(x), f_2(x), \ldots, f_o(x)]$, where o represents the number of optimization objectives (two objectives in this paper: execution makespan and economic cost). In workflow scheduling, each solution x includes two vectors: the first is a vector $(x_1, x_2, \ldots, x_n)$ of size n, where n represents the number of workflow tasks $(n = |A|)$ and $x_i = sched(A_i)$ denotes the resource executing task $A_i \in A$; the second is a permutation p of size n representing the execution order of all tasks.

For the workflow scheduling problem, simultaneously finding solutions that minimize both execution makespan and economic cost is impossible. To address this,

we introduce the concept of Pareto dominance to evaluate simultaneous optimality. If solution y has both lower makespan and lower cost than solution z, we say solution y dominates solution z. Conversely, if no mutual dominance exists between two solutions (one has lower makespan while the other has lower cost), the two solutions are non-dominated.

Using Figure 1 [Figure 1: see original paper] as an example, solution a dominates solution b because it has better makespan and cost. Similarly, solution a dominates solution c. Meanwhile, solutions a and d are non-dominated because solution a has better makespan while solution d has better cost. The set of optimal non-dominated solutions is called the Pareto front (the trend line containing solutions a, d, and e), representing a set of trade-off solutions among different objectives. Each solution in this set represents a different mapping scheme for workflow tasks with varying makespan and cost.

The method for measuring the quality of trade-off solutions is hypervolume. Given a set of trade-off solutions X, the hypervolume $HV(X)$ measures the enclosed region between points in X and a reference point W, as shown in Figure 1. The reference point is typically selected as the maximum objective values (i.e., highest makespan and economic cost). Therefore, more diverse and better points in X yield higher $HV(X)$ values. In Figure 1, the solution set containing points around the solid line is superior to the set containing points within the square, and thus the hypervolume value of the set containing all points around the solid line is higher than that of the set containing points within the square.

[Figure 1: see original paper]

---

4 Multi-objective Workflow Scheduling Algorithm Design

This section improves traditional single-objective optimization algorithms for heterogeneous computing environments (HEFT) and multi-objective optimization algorithms (MOHEFT) to make them applicable to Amazon EC2 cloud environments. In traditional heterogeneous systems, the quantity and types of resources are known. However, in cloud computing environments, although the total resource amount is finite, the resource provisioning model is on-demand and dynamic, with resource availability and computing capabilities being dynamically variable. To adapt the algorithm to the new cloud environment, we consider the resource input quantity as $m = N \times I$, where I represents the number of instance types provided by the cloud provider. This setting ensures that all combinations under the maximum resource quantity N are possible.

Algorithm 1 shows the execution process of the single-objective algorithm CHEFT. The design philosophy divides the task scheduling process into two phases: task ranking phase and task mapping phase. In the task ranking phase, a top-down ranking approach calculates each task' s distance to the

exit task in the workflow structure (represented by the maximum number of edges on all paths between the task and the exit task). Tasks are then sorted in descending order by their rank values to obtain the task scheduling priority sequence, ensuring that tasks with greater dependencies are prioritized for execution. In the task mapping phase, the scheduling solution with minimum makespan is sought across all resources. To meet cloud environment scheduling requirements, the algorithm tests whether a local scheduling scheme requires more than N resources; if not, it calculates makespan using the formula $TRank_i \leftarrow \max_{A_p \in pred(Rank_i)}\{TA_p + t(Rank_i, R_j)\}$. Otherwise, the makespan for that solution is set to infinity.

Algorithm 2 shows the execution process of the multi-objective algorithm CMO-HEFT. The design philosophy is: first, compute task rank values using a top-down ranking approach; then, initialize an empty set S that can accommodate K multi-objective trade-off solutions. The algorithm iteratively accesses the task set sorted in descending order by rank value to obtain the execution order. The objective is to continuously expand set S on m available resources to create and store new mapping solutions. The search process for new mappings continues until K solutions are found. Finally, solution quality is evaluated using Pareto dominance and hypervolume methods to obtain multi-objective optimal solutions.

Algorithm 1: CHEFT

```
Require: W=(A,D), A= _{i=1}^{n}{Ai}  // Workflow application
Require: N  // Maximum concurrent instances (resources)
Require: I  // Different instance types quantity
Require: R= _{j=1}^{m}{Rj}  // Resource set, m=N×I
Ensure: schedW={(Ai,sched(Ai))|Ai A}  // Workflow scheduling

function CHEFT(W,R)
    Rank←B-RANK(A)  // Rank tasks using B-rank
    schedW←  // Initialize workflow scheduling as empty set
    for i←1,n do  // Iterate through all ranked tasks
        Tmin←ϖ
        for j←1,m do  // Iterate through all resources
            if COUNTRESOURCES(schedW,m) N then  // If instance count is less than N
                TRank_i←max_{A_p pred(Rank_i)}{TA_p + t(Rank_i,Rj)}  // Compute completion t
            else
                TRank_i←ϖ  // Mark scheduling as invalid
            end if
            if TRank_i < Tmin then  // Save minimum completion time
                Tmin←TRank_i
                Rmin←Rj
            end if
        end for
        schedW←schedW (Rank_i,Rmin)  // Schedule task
```

```
    end for
    return schedW
end function
```

Algorithm 2: CMOHEFT

```
Require: W=(A,D), A= _{i=1}^{n}{Ai}  // Workflow application
Require: N  // Maximum concurrent instances (resources)
Require: I  // Different instance types quantity
Require: R= _{j=1}^{m}{Rj}  // Resource set, m=N×I
Require: K  // Number of trade-off solutions
Ensure:  _{i=1}^{K}{schedW_i}, schedW_i={(Ai,sched(Ai))|Ai A}  // K trade-off scheduling solu

function CMOHEFT(W,R,K)
    Rank←B-RANK(A)  // Rank tasks using B-rank
    S←  // Initialize empty set
    for k←1,K do  // Create K empty workflow scheduling solutions
        S_k←
    end for
    for i←1,n do  // Iterate through all ranked tasks
        S'←
        for j←1,m do  // Iterate through all resources
            for k←1,K do  // Iterate through all trade-off scheduling solutions
                s←S_k (Rank_i,Rj)  // Expand all intermediate scheduling schemes
                if COUNTRESOURCES(schedW,m) N then  // If instance count is less than N
                    // Compute makespan and cost
                else
                    Ts←ϖ  // Mark scheduling as invalid
                end if
                S'←S' {s}  // Add new mapping to all intermediate scheduling
            end for
        end for
        S'←SORTCROWDDIST(S',K)  // Sort by crowding distance
        S←FIRST(S',K)  // Select K solutions with highest crowding distance
    end for
    return S
end function
```

The difference between CHEFT and traditional single-objective algorithms lies in steps 17-20 of Algorithm 1. CMOHEFT checks resource quantity constraints at line 17; if not violated, makespan and cost are calculated as described previously. If violated, they are set to infinity, allowing the algorithm to discard solutions at line 24 and only generate trade-off solutions using at most N concurrent resources.

Algorithm Complexity Analysis: Given a workflow scheduling scenario with n tasks and m resources, the CHEFT algorithm first computes the rank value for each task, which has time complexity $O(n)$. It then iterates through the

sorted n tasks to find the scheduling solution with minimum makespan across m available resources, which has time complexity $O(n \times m)$. Therefore, CHEFT' s time complexity is $O(n) + O(n \times m) = O(n \times m)$.

CMOHEFT differs from CHEFT in two aspects: first, the creation method for multiple solutions in each iteration, and second, the consideration of possibilities for providing trade-off solutions. Both differences are reflected in lines 15-21 of CMOHEFT. Considering the trade-off solution set size is K, typically a constant smaller than n and m, CMOHEFT only requires K additional iterations compared to CHEFT. Therefore, CMOHEFT' s worst-case time complexity is $O(n \times m \times K)$.

---

5 Experimental Configuration

This section evaluates the effectiveness of the multi-objective algorithm CMOHEFT through simulation experiments using the WorkFlowSim [12] toolkit. We first present comparative metrics for evaluating algorithm performance quality, then describe the different types of test workflows and the configuration of Amazon EC2 facilities in the experiments. In addition to implementing CHEFT and CMOHEFT algorithms, we select the POSH algorithm [8] as a performance comparison baseline because it is also a multi-objective optimization algorithm that considers trade-off solutions.

5.1 Evaluation Metrics

We consider three metrics to compare the three algorithms: shortest workflow makespan, workflow execution economic cost, and hypervolume evaluation metric. The first two metrics evaluate single-objective optimization performance, while hypervolume evaluates the quality of multi-objective optimal trade-off solutions.

The specific hypervolume calculation method is: given a set of scheduling trade-off solutions X computed by the CMOHEFT algorithm, define $HV(X)$ as the hypervolume value measuring the enclosed region between individual points in solution set X (representing a scheduling solution) and a selected reference point W. Since workflow scheduling aims to simultaneously optimize execution makespan and cost, the reference point is selected as the point representing the highest makespan and execution cost in the trade-off solution set X. Individual solutions in the solution set are then sorted in descending order by their values on the first objective, and the hypervolume metric for the independent dominance region of individual solution p_i is calculated as:

$$HV(p_i) = |makespan(p_{i-1}) - makespan(p_i)| \times |cost(p_{i+1}) - cost(p_i)|$$

where $2 \leq i \leq K-1$, K represents the number of solutions in the solution set,

$makespan(p_i)$ denotes the makespan value of the i-th individual solution in the solution set, and $cost(p_i)$ is defined similarly.

5.2 Workflow Applications

The experiments introduce two types of workflow applications for testing: artificially configured multi-type synthetic workflows with different attributes, and real-world workflows from scientific domains.

1) Synthetic Workflows

A random workflow generator is used to produce three types of synthetic workflows (shown in Figures 2(a) and 2(b)), primarily to analyze the impact of task quantity on scheduling results. The defined workflow types can cover various workflow structures:

- **Type-1 Workflow:** Tasks can execute with parallelism degrees of 1 and 2.
- **Type-2 Workflow:** Tasks have higher execution parallelism, and the workflow is balanced (same number of tasks at each level).
- **Type-3 Workflow:** Tasks have higher execution parallelism, but the workflow is unbalanced (different number of tasks at each level).

Task sizes and data volumes follow a Gaussian distribution. For each workflow type, the number of tasks ranges between 100 and 1000, with 100 instances.

2) Real-world Workflow Applications

We consider two real-world workflow applications: WIEN2K and POV-Ray, with structures shown in Figures 2(c) and 2(d) [Figure 2: see original paper].

WIEN2K is a materials science workflow application that calculates solid electronic structures using density functional theory based on the full-potential linearized augmented plane wave method. It belongs to Type-2 workflow, consisting of two parallel sections executed by consecutive synchronization tasks. POV-Ray is an open-source tool for constructing three-dimensional graphics, widely used in biology, medicine, architecture, and mathematical visualization domains. It also belongs to Type-2 workflow.

[Figure 2: see original paper]

5.3 Resource Configuration

The maximum number of Amazon instances accessible to users is set to $N = 20$, with a total of five instance types as shown in Table 1, i.e., $I = 5$, $m = N \times I = 20 \times 5 = 100$. Meanwhile, the output data volume from Amazon to the external Internet is assumed constant and only occurs at workflow execution completion, thus not affecting scheduling results. In simulation experiments, costs for data transmission and reception are both zero, i.e., $PIR_i = 0$, $POR_i = 0$.

6 Results

6.1 Synthetic Workflows

1) Type-1 Workflow

Figure 3 [Figure 3: see original paper] shows the results of the three algorithms in terms of makespan, economic cost, and hypervolume performance. Figure 3(a) indicates that CMOHEFT' s hypervolume is superior to POSH across all evaluated instances. CHEFT is not included in the figure as it only yields a single solution with optimal makespan. Notably, for Type-1 workflows, CMOHEFT consistently maintains the same hypervolume value when computing scheduling solutions, indicating that the shape of the optimal trade-off solution set does not change with workflow scale. Regarding makespan (Figure 3(b)), all three algorithms obtain identical solutions, confirming that CMOHEFT' s performance does not degrade compared to CHEFT. Since POSH uses HEFT to obtain initial solutions, it also achieves the same makespan. Figure 3(c) shows that CMOHEFT and POSH achieve the same minimum-cost scheduling, with both algorithms using only m1.small instances throughout workflow execution, primarily due to the low parallelism of this workflow type. CHEFT, however, always incurs the highest cost.

Figure 3(d) illustrates the trade-off solutions of CMOHEFT and POSH, clearly showing that CMOHEFT produces higher-quality solutions. Specifically, CMOHEFT can reduce execution cost by approximately 45% on average with only about a 7% increase in time overhead, whereas POSH requires about a 25% increase in makespan to achieve the same cost reduction.

[Figure 3: see original paper]

2) Type-2 Workflow

Figure 4 [Figure 4: see original paper] shows the results for Type-2 workflows. Figure 4(a) demonstrates that CMOHEFT outperforms POSH in trade-off solution quality for all workflow scales. Different workflow scales result in Pareto fronts with different hypervolume values, indicating that the shape of the Pareto frontier in this problem depends on the number of tasks that can execute in parallel. For makespan (Figure 4(b)), CMOHEFT and POSH can sometimes achieve better makespan than CHEFT, primarily because CHEFT has greedy properties that make it prone to converging to local optima, while CMOHEFT and POSH overcome this limitation through larger search spaces. Regarding economic cost (Figure 4(c)), CMOHEFT and POSH achieve the same minimum cost with optimal makespan. The difference between CMOHEFT and POSH trade-off solutions in Figure 4(d) is more pronounced than in Type-1. Here, CMOHEFT reduces cost by about 30% with only a 1.4% increase in makespan, while POSH requires a 450% increase in makespan to achieve the same cost reduction.

[Figure 4: see original paper]

3) Type-3 Workflow

Figure 5 [Figure 5: see original paper] shows the execution results for Type-3 workflows, further confirming the validation results from the previous two workflow types. Overall, CMOHEFT is better suited for multi-objective workflow scheduling in Amazon EC2 cloud environments.

[Figure 5: see original paper]

6.2 Real-world Workflows

1) WIEN2k Workflow

[Content continues with real-world workflow evaluation···]

References

[1] Laili Y, Tao F, Zhang L, et, al. A study of optimal allocation of computing resources in cloud manufacturing systems [J]. International Journal of Advanced Manufacturing Technology, 2012, 63 (5): 671-690.

[2] Liu L, Zhang M, Lin Y, et al. A survey on workflow management and scheduling in cloud computing [C]// Proc of the 14th IEEE//ACM International Symposium on Cluster, Cloud and Grid Computing. [S. l.]: IEEE Press, 2014: 837-846.

[3] Rodriguez M, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds [J]. IEEE Trans on Cloud Computing, 2014, 2 (2): 222-235.

[4] Arabnejad H, Barbosa J G. A budget constrained scheduling algorithm for workflow applications [J]. Journal of Grid Computing, 2014, 12 (14): 1-15.

[5] Topcuoglu H, Hariri S, Wu MY, Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel Distributed Systems, 2012, 13 (3): 260-274.

[6] Zheng W, Sakellariou R. Budget-deadline constrained workflow planning for admission control [J]. Journal of Grid Computing, 2013, 11 (4): 633-651.

[7] 杨玉丽, 彭新光, 黄名选, 等, 基于离散粒子群优化的云工作流调度 [J]. 计算机应用研究, 2014, 31 (12): 3677-3681.

[8] Su S, Li J, Huang Q, et. al. Cost-efficient task scheduling for executing large programs in the cloud [J]. Parallel Computing, 2013, 39 (4): 177-188.

[9] Garg R, Singh A. Multi-objective workflow grid scheduling based on discrete particle swarm optimization [M]// Swarm, Evolutionary, and Memetic Computing. Berlin: Springer, 2012: 183-190.

[10] Garg R, Singh A K. Multi-objective workflow grid scheduling using -fuzzy dominance sort based discrete particle swarm optimization [J]. Journal of Supercomputing, 2014, 68 (2): 709-732.

[11] Zhang F, Cao J, Hwang K, et al. Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds [C]// Proc of the 3rd IEEE International Conference on Cloud Computing Technology and Science. Washington DC: IEEE Computer Society, 2012: 9-17.

[12] Chen W, Deelman E. WorkflowSim: a toolkit for simulating scientific workflows in distributed environments [C]// Proc of the 8th IEEE International Conference on e-Science. [S. l.]: IEEE Press, 2012: 1-8.

Note: Figure translations are in progress. See original paper for figures.