J. Parallel Distrib. Comput. 74 (2014) 2152–2165

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Multi-objective list scheduling of workflow applications in distributed computing infrastructures

Hamid Mohammadi Fard, Radu Prodan *, Thomas Fahringer

Institute for Computer Science, University of Innsbruck, Technikerstraße 21a, A-6020 Innsbruck, Austria

HIGHLIGHTS

- We propose a multi-objective scheduling framework for scientific workflows.
- We instantiate the framework for makespan, cost, energy, and reliability.
- We design a novel multi-objective list scheduling heuristic for workflows.
- We approximate the optimal solutions based on Pareto domination of user preferences.
- The solutions have better coverage compared to two related approaches.

ARTICLE INFO

Article history:
Received 16 June 2013
Received in revised form
30 October 2013
Accepted 5 December 2013
Available online 14 December 2013

Keywords:
Multi-objective scheduling
Scientific workflows
Distributed computing infrastructures

ABSTRACT

Executing large-scale applications in distributed computing infrastructures (DCI), for example modern Cloud environments, involves optimization of several conflicting objectives such as makespan, reliability, energy, or economic cost. Despite this trend, scheduling in heterogeneous DCIs has been traditionally approached as a single or bi-criteria optimization problem. In this paper, we propose a generic multi-objective optimization framework supported by a list scheduling heuristic for scientific workflows in heterogeneous DCIs. The algorithm approximates the optimal solution by considering user-specified constraints on objectives in a dual strategy: maximizing the distance to the user's constraints for dominant solutions and minimizing it otherwise. We instantiate the framework and algorithm for a four-objective case study comprising makespan, economic cost, energy consumption, and reliability as optimization goals. We implemented our method as part of the ASKALON environment (Fahringer et al., 2007) for Grid and Cloud computing and demonstrate through extensive real and synthetic simulation experiments that our algorithm outperforms related bi-criteria heuristics while meeting the user constraints most of the time.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

Scientific workflows emerged in the last decade as an attractive paradigm for programming large-scale applications in heterogeneous distributed computing infrastructures (DCI) such as Grids and Clouds. In this context, scheduling heterogeneous tasks including workflows is one of the traditional challenges in parallel and distributed computing. If we focus on the execution time also referred as makespan, the problem has been shown to be NP-complete, hence no polynomial algorithm for solving it exists (assuming $P \neq NP$). While this has been for decades the only optimization parameter of interest, modern DCIs are bringing along nowadays other parameters of equal importance such as reliability and economic cost (in Clouds), while recently energy consumption raises ever greater concerns too. Real-world scenarios are therefore confronted with a multi-objective optimization problem where many of these objectives are conflicting. For example, fast processors are typically rented by Cloud providers at higher prices, consume more energy, and may become unreliable due to the large user contention. In these scenarios, there is no single solution that optimizes all objectives, but a set of tradeoff solutions known as the Pareto frontier.

Until today, traditional scheduling researches [19] targeted makespan as the only optimization goal, while several isolated efforts addressed the problem by considering at most two objectives [17,21]. Although scheduling problems involve today multi-objective optimizations [14], a generic scheduling algorithm and framework for optimizing multiple conflicting objectives is still missing. Due to the NP-hard complexity of the makespan scheduling problem, practical approaches need to resort on heuristics to approximate the optimal solutions also in the multi-objective case. In this paper, we present a polynomial multi-objective algorithm for scientific workflow applications in heterogeneous DCIs that brings a two-fold novelty. First, we propose a general framework based on the multi-objective optimization theory for static

* Corresponding author.
E-mail addresses: hamid@dps.uibk.ac.at (H.M. Fard), radu@dps.uibk.ac.at, radu.prodan@uibk.ac.at (R. Prodan), tf@dps.uibk.ac.at (T. Fahringer).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.12.004

scheduling of scientific workflows in DCIs. We analyze and classify different objectives with respect to their impact on the optimization process and present a four-objective case study comprising makespan, economic cost, energy consumption, and reliability. Second, we support our framework through a list scheduling heuristic algorithm capable of dealing with more than two objectives (as restricted by related works). The algorithm uses constraints specified by the user for each objective and approximates the optimal solution by applying a dual strategy: maximizing the distance to the constraint vector for dominant solutions and minimizing it otherwise.

The paper is organized as the follows. In Section 2 we review the related work, followed by a short background in multi-objective optimization theory in Section 3. In Section 4, we formalize the abstract application, objective, and platform models underneath our approach. In Section 5, we instantiate this model by a case study comprising makespan, cost, reliability and energy as objectives. Section 6 presents a new multi-objective list scheduling heuristic for workflow applications, illustrated through a small example in Section 7. We extensively evaluate our method in Section 8 for real-world and synthetic workflows and conclude in Section 9.

2. Related work

Most related works are bi-objective approaches which we organized in three categories: unconstrained, single-constraint, and Pareto-based.

2.1. Unconstrained approaches

The two workflow scheduling heuristics (list and genetic-based) proposed in [7] tradeoff execution time for reliability. The algorithms consider no constraint and no weight for the objectives, and only concentrate on fairly optimizing the objectives. Scheduling of pipeline workflows on homogeneous platforms with respect to latency and throughput has been studied in [3]. The algorithm does not consider general workflows and assumes one constrained objective in each execution. In [1], a list workflow scheduling heuristic that considers makespan as the first objective and reliability as the second one has been proposed. The key idea is to replicate the activities on proper resources to gain both reliability and makespan optimization. This lexicographic method considers no constraint for the objectives and also assumes that the objectives are arranged in the order of their importance. The generic multi-objective approach in [8] targets task scheduling of different users in two situations: constant and time-invariant penalty functions. Each user tries to increase an own utility, while the scheduler is responsible for increasing the overall utility of the users by assigning them priorities. The algorithm is restricted to independent tasks.

2.2. Single-constraint approaches

Two algorithms called LOSS and GAIN [17] schedule a directed acyclic graph (DAG) under a budget constraint based on a two-phase optimization process: the first phase optimizes one criterion, while in the second phase applies a budget constraint. In [22], the authors solve a bi-criteria workflow scheduling problem by minimizing the execution cost while meeting a deadline. In the first step, they divide the workflow deadline into sub-deadlines for all activities. In the second step, they model the sequential activities as a Markov decision process solved using a value iteration method. Both papers consider only one constrained objective and try to optimize the other with respect to the defined constraint. Furthermore, they use a rescheduling phase that introduces significant overhead that makes the scheduling process non-scalable. The work in [21] proposes a bi-criteria genetic optimization algorithm that defines the fitness function as a combination of the partial fitness functions of the objectives. The algorithm is budget-constrained and considers no execution deadline. The algorithm in [2] targets DAG execution time minimization while keeping the number of failures equal to a constant x by replicating $x+1$ copies of each activity to different processors. The authors also suggest an extended algorithm which tries to improve the system reliability using redundancy.

2.3. Pareto-based approaches

The algorithm in [23] considers multi-objectives evolutionary algorithms for general workflow scheduling. The output is an approximation of the Pareto set and selecting the proper solution from this set remains a problem. The work in [11] is a similar approach for Grid workflow scheduling based on a multi-objective differential evolutionary algorithm that approximates the Pareto set by considering time and cost as objectives. In general, the major weakness of the evolutionary algorithms for scheduling problems is their slow convergence to good solutions, as we demonstrate in our experiments (Section 8). In contrast to Pareto-based approaches, our proposed algorithm can be categorized as an a-priori multi-objective scheduling method [14] looking for a single solution that satisfies the user's preferences and constraints. Our case study considers four objectives and uses Pareto relationships to find a solution that dominates or approaches the user constraints.

3. Multi-objective optimization background

We introduce several important concepts of the multi-objective optimization theory used in our method involving two steps: finding a set of optimal solutions and selecting the fittest solutions according to user preferences. We must distinguish two spaces as part of multi-objective optimization problems:

1. *solution* (or design) space X comprising all feasible solutions, for example the complete set of possible schedules of an application;
2. *objective* (or criterion) space O comprising an image of every element of X mapped to objective values.

In other words, each $x \in X$ maps to an image $o \in O$ which represents the objective values of x, as depicted in Fig. 1. A point $o \in O$ *dominates* $o' \in O$, denoted as $o \succ o'$, if o is not worse than o' with respect to all objectives and o is better for at least one of them. A point $o' \in O$ is said to be non-dominated if there is no $o \in O$ that dominates o'. A solution $x \in X$ is Pareto optimal (efficient) if its image in the objective space is non-dominated. The set of all Pareto-optimal solutions is called a *Pareto-optimal set*. The image set of all members of a Pareto-optimal set in the objective space is called a *Pareto frontier*. A vector comprising the best possible values for all objectives is called a *Utopia point* representing the ideal solution. Such a vector typically dominates the whole Pareto frontier and is therefore impossible to realistically achieve. A vector comprising the worst possible values for all existing objectives is called a *Nadir point*. The entire Pareto frontier dominates this point.

Let us assume that Fig. 1(b) depicts the objective space of the solution space of Fig. 1(a). The three solutions x, x' and x'' map to points o, o' and o'' in the objective space. If the optimization in this sample is minimizing both objectives O_1 and O_2, then the point o in the objective space is dominated by the point o'', while o'' has better values for both objectives O_1 and O_2. Two points o' and o'' are non-dominated points and consequently, the set $\{o', o''\}$ is the Pareto frontier. Therefore, the corresponding solutions x' and x'' are Pareto optimal and the set $\{x', x''\}$ is the Pareto set. The Utopia point is determined by selecting the best values of the objectives, as shown in Fig. 1(b). In contrast, the Nadir point is determined by selecting the worst values of the objectives. Typically, we need a single final solution selected by analysis of the preferences.

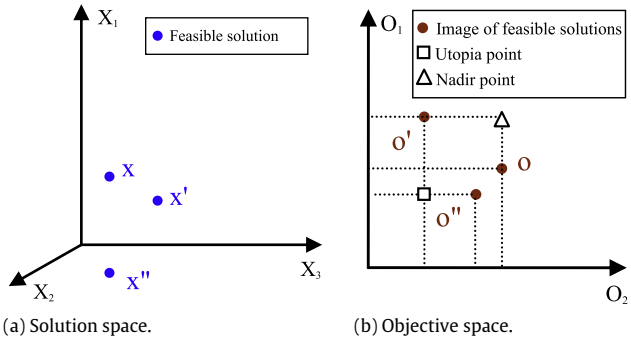


Fig. 1. A sample multi-objective problem.

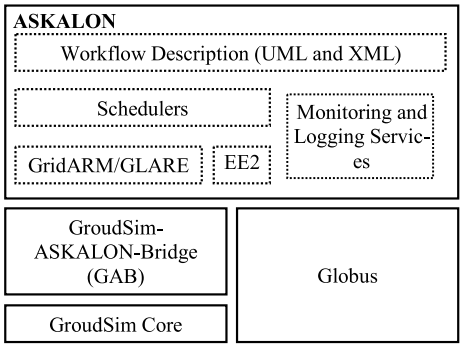


Fig. 2. Simplified architecture of the ASKALON environment.

4. Model

We define in this section the abstract application, multi-objective, platform, and execution models underneath our approach.

4.1. Application model

We model a *workflow application* as a DAG $W = (A, D)$ consisting of n *activities* $A = \bigcup_{i=1}^{n}\{A_i\}$ interconnected through control flow and data flow dependencies $D = \left\{\left(A_i, A_j, Data_{ij}\right) : \left(A_i, A_j\right) \in A \times A\right\}$, where $Data_{ij}$ represents the size of the data to be transferred to A_j from A_i before its execution. We use $pred(A_i)$ to denote the immediate *predecessor* set ($pred(A_i) = \{A_j \in A : \forall(A_j, A_i, Data_{ji}) \in D\}$) and $succ(A_i)$ to denote the immediate *successor* set of activity A_i($succ(A_i) = \{A_j \in A : \forall(A_i, A_j, Data_{ij}) \in D\}$). Each workflow has an activity with no predecessors called *entry* ($entry = A_i \in A : pred(A_i) = \emptyset$) and an activity with no successors called *exit* ($exit = A_i \in A : succ(A_i) = \emptyset$). We specify the computational workload of activity A_i in millions of instructions (MI), denoted as $work(A_i)$.

4.2. Multi-objective model

Let $(O_1, \ldots, O_k)$ denote the objective vector expressing a set of k *objectives* to be simultaneously optimized. Moreover, we assume the availability of a user-provided *constraint vector* $(C_1, \ldots, C_k)$ defining soft constraint values for the k objectives that shall be fulfilled in a best-effort fashion (in contrast to hard constraints which must be fulfilled in all circumstances [13]). For instance, $(O_1, O_2) = (Makespan, Reliability)$ indicates that the scheduling objectives are makespan and reliability and $(C_1, C_2) = (1000, 0.98)$ specifies a constraint of 1000 for makespan and of 0.98 for reliability. To assist the user in specifying feasible constraints, the scheduler computes and proposes a *validity range* $\left[O_i^{(\min)}, O_i^{(\max)}\right]$ for each objective $i \in \{1..k\}$ from which the user can freely select. In Section 5, we will illustrate how to compute these ranges for different classes of objectives. Choosing the best value for each objective from its range represents the Utopia point that is usually impossible to achieve. Additionally, the user also specifies a *weight vector* $(w_1, \ldots, w_k)$ defining the contribution of each objective in the decision making process (the largest weight has the most significant impact on the final solution). If the user is not able to define the constraint and weight vectors, the algorithm uses the Utopia point as the constraint vector with equal weights for objectives. Since for each multi-objective problem we have a Pareto set of optimal solutions, the constraint and weight vectors have a direct impact on the position of the solution in the solution space. In other words, different constraint and weight vectors conduct the algorithm toward different optimal solutions.

4.3. Platform model

We consider the hardware platform as a set $R = \bigcup_{j=1}^{m} R_j$ of m heterogeneous resources. Each resource R_j is characterized by a *property vector* $(v_{1j}, \ldots, v_{kj})$, which we will define in Section 5 for our four-objective case study. We assume the resources to be connected by a full mesh network, where each point-to-point connection has a different bandwidth. We further assume that workflow activities have dedicated access to bare metal resources with no virtualization or other form of external load involved, and are non-preemptive (i.e. it is not allowed to suspend an activity and resume it later). We also assume the possibility of reserving resources in advance.

We define a *schedule* for a workflow $W = (A, D)$ as a mapping:

$$sched : A \rightarrow R \tag{1}$$

from the set of n activities to the set of m resources. The solution space X has an exponential cardinality: $|X| \geq m^n$. If we do not discriminate among different combinations of X, then $|X| = m^n$.

4.4. Execution model

We designed our method as part of the ASKALON environment [10] which supports the development and execution of scientific workflows on Grid and Cloud infrastructures. As shown in Fig. 2, ASKALON supports high-level abstractions for programming complex scientific applications by using either a textual XML representation or a graphical UML-based diagram. Users can select between a set of different schedulers that interact with resource management (GridARM/GLARE), prediction, and enactment engine (EE2) services to discover resources, estimate execution times of activities, and execute workflows. Monitoring and logging services store the history of the executions in a data repository. Since simulation helps researchers analyze and further optimize the behavior of software systems, we also interfaced ASKALON at the backend with the GroudSim simulator [15] that supports modeling of Grid and Cloud computational and network resources, including job submissions, file transfers, failure models, background load, and cost models.

5. Four-objective case study

Based on the generic model defined in Section 4, we address a case study consisting of four objectives important for real-life applications: *makespan* (objective O_1), *economic cost* (O_2), *energy consumption* (O_3), and *reliability* (O_4). Among them, energy consumption is an objective of the resource provider, while the other three are concerns of the users. These objectives differ in several important classification criteria [20] summarized in Table 1, where m and n represent the number of resources and activities.

Table 1
Objective function classification based on [20].

Objective	Structural dependency	Aggregation	Complexity	Direction
Makespan	Structure-dependent	Additive	$O(m^n)$	Minimize
Economic cost	Structure-independent	Additive	$O(m \cdot n)$	Minimize
Energy	Structure-independent	Additive	$O(m \cdot n)$	Minimize
Reliability	Structure-independent	Multiplicative	$O(m \cdot n)$	Maximize

5.1. Makespan

The execution time of an activity A_i on resource R_j is the sum of the longest input transfer time (from all inputs to A_i) and the activity computation time:

$$O_1\left(A_i, R_j\right) = \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{jq}} \right\} + \frac{work(A_i)}{v_{1j}}, \tag{2}$$

where $sched(A_p) = R_q$, (i.e. A_p is scheduled on R_q), b_{jq} is the transfer bandwidth between resources R_q and R_j, and v_{1j} is the computation speed of R_j in millions of instructions per second (MIPS). We recursively compute the completion time or makespan of an activity A_i on a resource R_j as follows:

$$end\left(A_i, R_j\right) = \begin{cases} O_1\left(A_i, R_j\right), & A_i = entry; \\ \max\limits_{\forall A_p \in pred(A_i)} \left\{ end\left(A_p, sched(A_p)\right) \right. \\ \qquad \left. + O_1\left(A_i, R_j\right) \right\}, & A_i \neq entry. \end{cases} \tag{3}$$

Thus, the workflow makespan is the longest completion time of its activities:

$$O_1(W) = \max_{i \in \{1..n\}} \left\{ end\left(A_i, sched(A_i)\right) \right\}. \tag{4}$$

To estimate a feasible validity range $C_1 \in \left[O_1^{(\min)}, O_1^{(\max)}\right]$, we use the HEFT algorithm [19] for the lower bound $O_1^{(\min)}$ (an NP-complete problem), and the sequential execution of activities on the slowest computing machine for $O_1^{(\max)}$.

5.2. Economic cost

For the economic cost, we use a "pay-as-you-go" pricing model common in commercial Clouds such as Amazon that charge customers a fixed price per time unit of resource usage. The economic cost objective O_2 in our model consists of four components: (1) computation cost, (2) storage cost, (3) incoming data transfer, and (4) outgoing data transfer costs. Each resource R_j is charged a fixed cost per time unit $v_{2j}^{(p)}$ for each cost component $p \in \{1..4\}$. Therefore, the economic cost is the sum of computation, data storage, and data transfer costs:

$$\begin{aligned} O_2\left(A_i, R_j\right) = {} & O_1\left(A_i, R_j\right) \cdot v_{2j}^{(1)} + TotalData(A_i) \\ & \cdot O_1\left(A_i, R_j\right) \cdot v_{2j}^{(2)} + Input'(A_i) \cdot v_{2j}^{(3)} \\ & + Output'(A_i) \cdot v_{2j}^{(4)}, \end{aligned} \tag{5}$$

where $TotalData(A_i)$ is the total data storage required for executing the activity A_i including input, output and temporary data, and $Input'(A_i)$ ($Output'(A_i)$) is the sum of the input data sizes transferred to (from) A_i from (to) activities executed on resources other than R_j. Internal data transfers in Clouds are usually free of cost. Economic cost is an additive structure-independent objective that can be computed for a workflow as:

$$O_2(W) = \sum_{i=1}^{n} O_2\left(A_i, sched(A_i)\right). \tag{6}$$

The validity range $C_2 \in \left[O_2^{(\min)}, O_2^{(\max)}\right]$ can be calculated using a greedy algorithm by simply mapping each activity onto the cheapest, respectively the most expensive resource in terms of its price and speed ratio.

5.3. Energy consumption

We calculate the energy consumption of an activity A_i on a resource R_j by multiplying its makespan by the power consumption v_{3j}, modeled as in [16,12]:

$$O_3\left(A_i, R_j\right) = O_1\left(A_i, R_j\right) \cdot v_{3j}. \tag{7}$$

We neglect the power consumption of data transfers (lying in the range [2 W; 5 W]), which is much lower compared to today's most efficient processors (consuming about 40 W) and high-end servers (consuming between [100 W, 300 W] in idle and over 500 W in fully loaded mode). The workflow energy consumption is also an additive structure-independent criterion that we calculate as follows:

$$O_3(W) = \sum_{i=1}^{n} O_3\left(A_i, sched(A_i)\right). \tag{8}$$

Energy consumption can be optimally solved by mapping all activities onto the machine with the least energy consumption with respect to its computational speed. This value represents the lower bound $O_3^{(\min)}$ of the third constraint $C_3 \in \left[O_3^{(\min)}, O_3^{(\max)}\right]$, where $O_3^{(\max)}$ is again calculated by scheduling each activity on the most energy consuming resource.

5.4. Reliability

We assume resource failures to be statistically independent and follow a constant failure rate v_{4j} for each resource $j \in [1, m]$. We consider the reliability of an activity A_i as the probability of successful completion on a resource R_j, modeled based on previous work using an exponential distribution [7,2]:

$$O_4\left(A_i, R_j\right) = e^{-v_{4j} \cdot O_1\left(A_i, R_j\right)}. \tag{9}$$

Since a workflow is reliable if all its activities are reliable, we define its probability of success as the product of success probabilities of all activities:

$$O_4(W) = \prod_{i=1}^{n} e^{-v_{4j} \cdot O_1\left(A_i, sched(A_i)\right)}. \tag{10}$$

In terms of the validity range $C_4 \in \left[O_4^{(\min)}, O_4^{(\max)}\right]$, reliability is also a structure-independent objective whose lower bound $O_4^{(\min)}$ is the multiplication of the reliability of the most unreliable resource for each activity, while the upper bound $O_4^{(\max)}$ is the multiplication of the reliability of the most reliable resource for each activity. A final peculiarity of reliability is its optimization direction that should be maximized, as opposed to minimizing the other objectives.

5.5. Utopia and Nadir points

In this four objective case study, a schedule S represents an Utopia point if it simultaneously minimizes makespan, economic cost, and energy consumption ($O_i(S) = O_i^{(\min)}(S)$, $i \in [1, 3]$), and maximizes reliability ($O_4(S) = O_4^{(\max)}(S)$). Conversely, a Nadir point maximizes makespan, cost and energy, and minimizes reliability. It is important to mention that, from these four objectives, only computing the minimum makespan is an NP-complete problem approximated by the HEFT algorithm. The rest of the objectives can be minimized or maximized by serializing the workflow on the most expensive/cheap, energy consuming/saving, or (un-)reliable machine, as explained in the previous subsections.

$$C_2^{(i)} = C_2 \cdot \frac{C_1^{(i)} \cdot \alpha_{avg} + TotalData(A_i) \cdot C_1^{(i)} \cdot \beta_{avg} + Input(A_i) \cdot \rho_{avg} + Output(A_i) \cdot \varrho_{avg}}{\sum_{j=1}^{n} (C_1^{(j)} \cdot \alpha_{avg} + TotalData(A_j) \cdot C_1^{(j)} \cdot \beta_{avg} + Input(A_j) \cdot \rho_{avg} + Output(A_j) \cdot \varrho_{avg})}, \quad (11)$$

Box I.

Algorithm 1: MOLS algorithm.

Input: Workflow application: $W = (A, D)$; Objective vector: $(O_1, \ldots, O_k)$; Weight vector: $(w_1, \ldots, w_k)$; Resource set: $R = \bigcup_{j=1}^{m} R_j$
Output: Schedule of W: $sched_W = \bigcup_{i=1}^{n} (A_i, sched(A_i))$

1 **begin**
  /* Phase one: constraint vector partitioning */
2 **for** $i \leftarrow 1$ **to** k **do**
3 Compute validity ranges by applying Eqs. 4–10: $\left[O_i^{(\min)}, O_i^{(\max)}\right]$
4 Select constraints: $C_i \in \left[O_i^{(\min)}, O_i^{(\max)}\right]$ /* select best value, if user indicates no constraints */
5 **end**
6 **for** $i \leftarrow 1$ **to** n **do**
7 Compute partial constraint vector: $\left[C_1^{(i)}, \ldots, C_k^{(i)}\right]$
8 **end**
  /* Phase two: activity ordering */
9 $B \leftarrow Sort(A, B\text{-}ranks)$
  /* Phase three: activity mapping */
10 $sched_W \leftarrow \emptyset$
11 **for** $i \leftarrow 1$ **to** n **do**
12 Compute intermediate constraint vector: I_i
13 Generate intermediate schedules: $M \leftarrow \bigcup_{j=1}^{m} (sched_W \cup (B_i, R_j))$
14 Select schedules that dominate I_i: $D_1 \leftarrow \{S \in M : O(S) \succ I_i\}$
15 **if** $D_1 \neq \emptyset$ **then**
16 $sched_W \leftarrow S \in D_1 : d(O(S), I_i) \geq d\left(O\left(S'\right), I_i\right), \forall S' \in D_1$, where:
17 $d(O(S), I_i) = \sqrt{\sum_{j=1}^{k} w_i \cdot \left[N\left(O_j(S)\right) - N\left(I_{ji}\right)\right]^2}$ (see Eq. 22)
18 **else**
  /* there exist no dominating solutions */
19 Select closest schedule to I_i:
  $T_i \leftarrow S \in M : d(O(S), I_i) \leq d\left(O\left(S'\right), I_i\right), \forall S' \in M$
20 Select schedules that dominate T_i: $D_2 \leftarrow \{S \in M : O(S) \succ T_i\}$
21 **if** $D_2 = \emptyset$ **then**
22 Select furthest schedule to T_i:
  $sched_W \leftarrow \left\{S \in D_2 \mid \forall S' \in D_2, d(O(S), T_i) \geq d\left(O(S'), T_i\right)\right\}$
23 **else**
24 Select T_i as solution: $sched_W \leftarrow T_i$
25 **end**
26 **end**
27 **end**
28 **return** $sched_W$
29 **end**

Algorithm 2: Makespan constraint partitioning algorithm.

Input: Workflow application $W = (A, D)$; Makespan constraint: C_1
Output: Makespan constraint $C_1^{(i)}$ of activity A_i, $\forall i \in [1, n]$:

1 **begin**
2 $pathSet \leftarrow$ set of all path from the first to the last activity
3 **while** $pathSet \neq \emptyset$ **do**
4 $CP \leftarrow$ longest path in $pathSet$
5 $pathSet \leftarrow pathSet \setminus CP$
6 $assigned \leftarrow \left\{A_i \in CP : C_1^{(i)} is\ calculated\right\}$
7 $unassigned \leftarrow \{A_i \in CP : A_i \notin assigned\}$
8 $spentTime \leftarrow \sum_{A_i \in assigned} C_1^{(i)}$
9 $unassignedLoad \leftarrow \sum_{A_i \in unassigned} work(A_i)$
10 **foreach** $A_i \in unassigned$ **do**
11 $C_1^{(i)} \leftarrow \frac{C_1 - spentTime}{unassignedLoad} \cdot work(A_i)$
12 **end**
13 **end**
14 **end**

6. Multi-objective list scheduling algorithm

We propose a *multi-objective list scheduling* (MOLS) algorithm that attempts to find a solution which dominates or which converges to the constraint vector. The MOLS algorithm is based on three phases, outlined in pseudo-code in Algorithm 1. In the first phase (lines 2–8), it partitions the constraint vector across all workflow activities according to their workload. In other words, this phase estimates the objectives' sub-constraints for each individual activity using the constraint vector. In the second phase (line 9), it assigns a rank to each workflow activity and sorts them in ascending order. In the third phase (lines 10–26), the algorithm attempts to allocate the most appropriate resource to each activity with respect to the estimated sub-constraints. In the following, we describe each phase in detail and give a small illustrative example in Section 7 for a better understanding.

6.1. Phase one: constraint vector partitioning

In the first phase, we compute the validity ranges of each objective O_i (line 3) and select a constraint C_i for each O_i (line 4). Afterwards, we partition the constraint vector $(C_1, \ldots, C_k)$ among the workflow activities to create a partial constraint vector $\left(C_1^{(i)}, C_2^{(i)}, \ldots, C_k^{(i)}\right)$ for each activity A_i (lines 6–8). In other words, each $C_j^{(i)}$ in this vector represents an estimated constraint of the objective O_j for activity A_i. Creating the partial constraint vectors is strictly dependent on the nature of the objectives. We describe next the details for computing the vector $\left(C_1^{(i)}, C_2^{(i)}, C_3^{(i)}, C_4^{(i)}\right)$ in our four-objective case study.

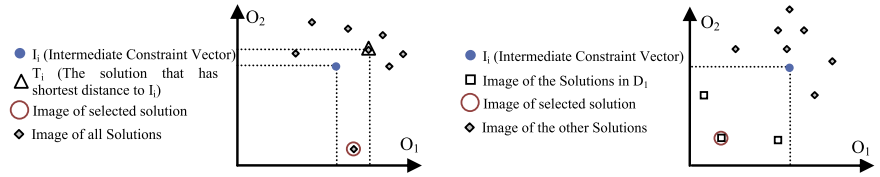
First, the makespan constraint is the most challenging objective for partitioning since it is a structure-dependent objective with no linear relation between the overall workflow makespan and the individual activities. To accurately partition a makespan constraint, Algorithm 2 creates first in line 2 the set of all paths from the first to the last activity. In line 4, we assign the longest path in this set to the variable *CP*, where the length is calculated by adding the workloads of all activities in the path. In line 5, we remove *CP* from the *pathSet* and build two sets: *assigned* is the set of activities in *CP* whose time constraints are already calculated (line 6) and *unassigned* is the rest (line 6). In line 8, we calculate the sum of the time constraints of the *assigned* activities and in line 9 the total workload of the *unassigned* ones. We use these values in line 11 to calculate the time constraints of all the unassigned activities in *CP* and repeat the process for all paths in *pathSet*. We trace this algorithm using an example in Section 7 for a better understanding.

Second, the contribution of the cost constraint of each activity is proportional to its cost compared to the cost of the entire workflow averaged over all resources, as described by Eq. (11) given in Box I where α_{avg} is the average computation cost, β_{avg} is the average storage cost, ρ_{avg} and ϱ_{avg} are the average incoming and outgoing data transfer costs across all resources, and $Input(A_i)$ and $Output(A_i)$ are the aggregated input and output data sizes processed by activity A_i.

Third, we consider energy consumption as a linear function of time that we partition as follows:

$$C_3^{(i)} = C_3 \cdot \frac{work(A_i)}{\sum_{j=1}^{n} work(A_j)}. \quad (12)$$

Finally, we partition the reliability constraint using two abstract parameters: average DCI failure rate λ_{DCI} and average DCI speed



(a) $D_1 \neq \emptyset$. (b) $D_1 = \emptyset$.

Fig. 3. Possible solutions in a sample two-dimensional objective space.

υ_{DCI}. Similar to Eq. (9) [7,2], we estimate the partial reliability constraint of activity A_i as:

$$C_4^{(i)} = e^{-\lambda_{DCI} \cdot \frac{work(A_i)}{\upsilon_{DCI}}}. \tag{13}$$

Since reliability is a multiplicative objective (see Section 5.4), we have:

$$C_4 = \prod_{i=1}^{n} C_4^{(i)} = \prod_{i=1}^{n} e^{-\lambda_{DCI} \cdot \frac{work(A_i)}{\upsilon_{DCI}}} = e^{-\frac{\lambda_{DCI}}{\upsilon_{DCI}} \cdot \sum_{i=1}^{n} work(A_i)}. \tag{14}$$

After applying the natural logarithm function to both sides of this equation, the ratio of the abstract parameters is: $\frac{\lambda_{DCI}}{\upsilon_{DCI}} = -\frac{\ln(C_4)}{\sum_{i=1}^{n} work(A_i)}$. By substituting this ratio in the $C_4^{(i)}$ definition, we obtain:

$$C_4^{(i)} = e^{\ln(C_4) \cdot \frac{work(A_i)}{\sum_{j=1}^{n} work(A_j)}}. \tag{15}$$

6.2. Phase two: activity ordering

In the second phase of the Algorithm 1 (line 9), we sort the workflow activities ascending order according to their bottom level (B-rank), defined in the graph theory for each activity A_i as the longest path to the last (*exit*) activity, including the activity A_i itself:

$$rank(A_i) = \begin{cases} work(A_i) + \max\limits_{A_j \in succ(A_i)} \{Data_{ij} \\ \quad + rank(A_j)\}, \quad A_i \neq exit; \\ work(A_i), A_i = exit. \end{cases} \tag{16}$$

We save this sorted ranking into a new list B that determines the scheduling order of activities in the third phase. The reason for using the makespan for ordering the workflow activities is twofold: (1) it is the only structure-dependent objective that preserves the precedence relationships in the ordered list and (2) it is the objective with the largest impact on the other three objectives.

6.3. Phase three: activity mapping

In the mapping phase (lines 10–26), we traverse the activities from the ordered list B and schedule them onto the "best" resource according to the k objectives. We initialize the scheduling solution with the empty set (line 10) and, at each iteration $i \in [1; n]$, we incrementally add a new mapping $(B_i, sched(B_i))$ until we build the final schedule:

$$sched_W = \bigcup_{j=1}^{n} \left(B_j, sched(B_j)\right). \tag{17}$$

Let $I_i = (I_{1i}, \ldots, I_{ki})$ denote the intermediate constraint vector at mapping step i (line 12) representing a sub-constraint vector for the sub-workflow comprising the activities $B_1, \ldots, B_i$. In the four-objective case study, we calculate I_i using the activities' partial constraint vectors, determined as defined in Section 5 for each objective. At each iteration $i \in [1; n]$, we store all m possible solutions for scheduling the activity B_i in the set of intermediate schedules (line 13):

$$M = \bigcup_{j=1}^{m} \left(sched_W \cup \left(B_i, R_j\right)\right), \tag{18}$$

where m is the number of resources. To find the "best" resource in each iteration i, we select all schedules from M whose objectives dominate the intermediate constraint vector I_i in a set D_1 (line 14):

$$D_1 = \{S \in M : O(S) \succ I_i\}, \tag{19}$$

where $O(S)$ represents the values of the objectives of the solution S. If $D_1 \neq \emptyset$ (see Fig. 3(a)), we select the member of this set with the longest weighted Euclidean distance to the intermediate constraint I_i (line 17):

$$d\left(O(S), I_i\right) = \sqrt{\sum_{j=1}^{k} w_j \cdot \left[N\left(O_j(S)\right) - N\left(I_{ji}\right)\right]^2}, \tag{20}$$

since we want to improve our solution on all objectives simultaneously (line 28):

$$sched_W = S \in D_1 : d\left(O(S), I_i\right) \geq d\left(O\left(S'\right), I_i\right), \quad \forall S' \in D_1. \tag{21}$$

Since different objectives are usually incomparable, we normalize them using the following normalization function [14]:

$$N : \left[O^{(\min)}, O^{(\max)}\right] \to [0, 1], \qquad N(x) = \frac{x - O^{(\min)}}{O^{(\max)} - O^{(\min)}}. \tag{22}$$

Therefore, $N\left(O_j(S)\right)$ and $N\left(I_{ji}\right)$ represent the j-th normalized objective values of S and I_i, respectively. If there are no solutions in D_1 that dominate the intermediate constraint vector ($D_1 = \emptyset$, see Fig. 3(b)), we first select a solution called T_i with the shortest weighted distance to I_i (line 19):

$$T_i = S \in M : d\left(O(S), I_i\right) \leq d\left(O\left(S'\right), I_i\right), \quad \forall S' \in M, \tag{23}$$

and then generate the set D_2 comprising the solutions in M that dominate T_i (line 20), similar to the previous case (see Eq. (19)). If $D_2 \neq \emptyset$, we choose the solution with the longest distance to T_i (line 22) as the final solution. If $D_2 = \emptyset$, we choose T_i as the final solution (line 24). After all activities have been scheduled, the algorithm returns $sched_W$ as the final solution (line 28).

6.4. Pareto optimality

Theorem 1. *The schedule defined in Eq. (21) is Pareto optimal.*

Proof. We disprove that there is no solution in D_1 able to dominate $sched_W$. Suppose Fig. 4 is the objective space of a sample bi-objective problem, where $D_1 = \left\{o, o', o''\right\}$ presents the set of all solutions that dominate the intermediate constraint vector I_i. The point $o \in D_1$ is the solution with the longest Euclidean distance to I_i. If o is not Pareto optimal, then there exist another point x that dominates o. Since x is dominating o, it must reside in the

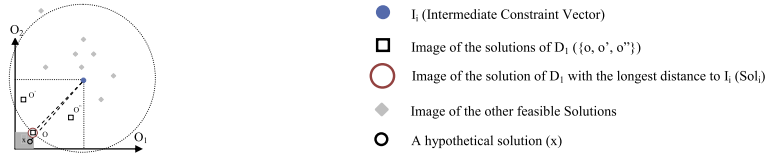
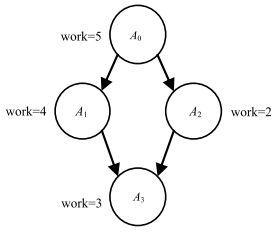


Fig. 4. Pareto optimality of the solution with the longest Euclidean distance.



(a) Sample workflow.

Resource property	R_0	R_1
Speed (MIPS)	$v_{10} = 1$	$v_{11} = 2$
Economic cost	$v_{20} = 10$	$v_{21} = 15$

(b) Sample DCI.

Fig. 5. Illustrative example.

gray rectangle highlighted in Fig. 4. Thus, x lies outside the circle with the center of I_i and the radius equal to the distance between I_i and o. Consequently, the distance between I_i and o is less than the distance between I_i and x which contradicts the assumption that o has the longest distance to I_i. ■

6.5. Complexity

Theorem 2. *The time complexity of the four-objective MOLS algorithm for a workflow with n activities and a DCI with m resources is* $\mathcal{O}\left(m \cdot n^3\right)$.

Proof. In the first phase of the algorithm, the most time consuming step in the loop between lines 2–4 is computing the makespan's lower bound, which is $\mathcal{O}\left(m \cdot n^2\right)$ [19] in case of HEFT. In computing the partial constraint vectors (lines 6–8), the most complex part is again related to the makespan structure-dependent objective which consumes $\mathcal{O}(n+e)$ in Algorithm 2, where e is the number of workflow dependencies. Since the maximum number of edges in a DAG is $\frac{n\cdot(n-1)}{2}$, this gives us a complexity of $\mathcal{O}\left(n^2\right)$. Therefore, the time complexity of the MOLS's first phase is $\mathcal{O}\left(m \cdot n^2\right)$. In the second phase, ranking activities has a complexity of $\mathcal{O}(n+e)$ or $\mathcal{O}(n^2)$ in the worst case. Since sorting the list B of activities (line 9) also has a time complexity of $\mathcal{O}\left(n^2\right)$ in the worst case, the time complexity of the second phase is $\mathcal{O}\left(n^2\right)$. In the third phase, the time complexity of the ith loop iteration is $\mathcal{O}\left(m \cdot i^2\right)$ because of the intermediate constraint vector I_i computation in line 12. Thus, the time complexity of the unrolled loop is $\sum_{i=1}^{n} m \cdot i^2$. Since $\sum_{i=1}^{n} i^2 = \frac{n\cdot(n+1)\cdot(2n+1)}{6}$, the third phase has a complexity of $\mathcal{O}\left(m \cdot n^3\right)$. Since the time complexity of the third phase is dominant, the complexity of the MOLS algorithm is $\mathcal{O}\left(m \cdot n^3\right)$. ■

As already described, makespan is the objective with the highest impact on the time complexity of the algorithm. Removing the makespan decreases the complexity of the algorithm to $\mathcal{O}\left(m \cdot n^2\right)$, while removing the other three objectives brings no considerable reduction.

7. Example

In this section, we present an example that illustrates the individual steps of the MOLS algorithm for a better understanding. Suppose we have the workflow depicted in Fig. 5(a) and the DCI summarized in Fig. 5(b). For simplicity of the calculations, we only consider two objectives: makespan (O_1) and cost (O_1) with equal weights of one ($w_1 = w_2 = 1$) and no data transfer between activities.

Table 2
Iteration steps of Algorithm 2 for the example in Fig. 5.

Line	First iteration	Second iteration
4	$CP = (A_0, A_1, A_3)$	$CP = (A_0, A_2, A_3)$
6	$assigned = \emptyset$	$assigned = \{A_0, A_1, A_3\}$
7	$unassigned = \{A_0, A_1, A_3\}$	$assigned = \{A_2\}$
8	$spentTime = 0$	$spentTime = 3 + 5 = 8$
9	$unassignedLoad = 12$	$unassignedLoad = 2$
11	$C_1^{(0)} = \frac{12-0}{12} \cdot 5 = 5$	$C_1^{(2)} = \frac{12-8}{2} \cdot 2 = 4$
11	$C_1^{(1)} = \frac{12-0}{12} \cdot 4 = 4$	–
11	$C_1^{(3)} = \frac{12-0}{12} \cdot 3 = 3$	–

Table 3
Partial constraints and activity ranks for the example in Fig. 5.

i	Activity	$C_1^{(i)}$	$C_2^{(i)}$	Partial constraint vector	Ranks
0	A_0	5	$\frac{600}{14} \cong 43$	(5, 43)	3+4+5 = 12
1	A_1	4	$\frac{480}{14} \cong 34$	(4, 34)	3 + 4 = 7
2	A_2	4	$\frac{240}{14} \cong 17$	(4, 17)	3 + 2 = 5
3	A_3	3	$\frac{360}{14} \cong 26$	(3, 26)	3

First of all, we calculate the validity range of each objective, as described in Section 5.1. The makespan lower bound is the solution delivered by HEFT, which is in this case: $O_1^{(\min)} = 6$. The upper bound is the sequential execution of the activities on the slowest R_0 resource: $O_1^{(\max)} = \frac{5+4+2+3}{1} = 14$. Therefore, $C_1 \in [6, 14]$. The cost lower bound corresponds to mapping each activity onto the cheapest R_0 resource: $O_2^{(\min)} = (5+4+2+3) \cdot \frac{15}{2} = 105$, while the upper bound maps each activity onto the most expensive R_1 resource: $O_2^{(\max)} = (5+4+2+3) \cdot \frac{10}{1} = 14$. Thus, $C_2 \in [105, 140]$. From the calculated validity ranges, we assume that the user specifies the constraint vector, e.g.: $(C_1, C_2) = (12, 120)$.

In the first phase, the algorithm calculates the partial constraint vectors. For makespan constraint partitioning, we use Algorithm 2 which iterates twice in the outer while loop since $pathSet = \{(A_0, A_1, A_3), (A_0, A_2, A_3)\}$. A trace of the algorithm is summarized in Table 2. Cost constraints are partitioned in an easier way as defined in Section 6.1, for example: $C_2^{(1)} = \frac{5\cdot 120}{5+4+2+3} \cong 43$. The makespan and cost partial constraints are shown in the third and fourth columns of Table 3.

In the second phase, we sort the activities in list B (based on their B-rank) as shown in the last column of Table 3, yielding $[A_0, A_1, A_2, A_3]$.

Table 4
Iteration steps of the mapping phase for the example in Fig. 5.

i	M	I_i	$O(M)$	D_1	$sched_W$
0	$\{(A_0,R_0),(A_0,R_1)\}$	(5, 43)	$\{(5,50),(2.5,37.5)\}$	$\{(A_0,R_1)\}$	$\{(A_0,R_1)\}$
1	$\{\{(A_0,R_1),(A_1,R_0)\},\{(A_0,R_1),(A_1,R_1)\}\}$	(9, 77)	$\{(6.5,77.5),(4.5,67.5)\}$	$\{\{(A_0,R_1),(A_1,R_1)\}\}$	$\{(A_0,R_1),(A_1,R_1)\}$
2	$\{\{(A_0,R_1),(A_1,R_1),(A_2,R_0)\},\{(A_0,R_1),(A_1,R_1),(A_2,R_1)\}\}$	(9, 94)	$\{(4.5,87.5),(5.5,82.5)\}$	$\{\{(A_0,R_1),(A_1,R_1),(A_2,R_0)\},\{(A_0,R_1),(A_1,R_1),(A_2,R_1)\}\}$	$\{(A_0,R_1),(A_1,R_1),(A_2,R_0)\}$
3	$\{\{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_0)\},\{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_1)\}\}$	(12, 120)	$\{(7.5,117.5),(6,110)\}$	$\{\{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_0)\},\{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_1)\}\}$	$\{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_1)\}$

In the third phase, we iterate over the sorted activity list and find an appropriate mapping of each activity in every iteration (see Table 4). At iteration zero, we create the set D_1 of solutions in $M = \{(A_0,R_0),(A_0,R_1)\}$ whose objective values dominate the intermediate constraint vector $I_0 = (5, 43)$. The objective space of M is: $O(M) = \{(5, 50), (2.5, 37.5)\}$. Out of this solution set, only the second element (2.5, 37.5) dominates I_0, thus $D_1 = \{(A_0, R_1)\}$. Because D_1 has only one member, we have $sched_W = (A_0, R_1)$. The second iteration of the algorithm operates similarly, as summarized in Table 4 in the row $i = 1$. In the third iteration ($i = 2$), we create the set D_1 of solutions in $M = \{\{(A_0, R_1), (A_1, R_1), \{(A_0, R_1), t(A_2, R_0)\}, (A_1, R_1), (A_2, R_1)\}\}$ whose objective values dominate the intermediate constraint vector $I_2 = (9, 94)$. The objective space of M is: $O(M) = (4.5, 87.5), (5.5, 82.5)$. Since both members of $O(M)$ dominate the vector I_2, then:

$$D_1 = \{\{(A_0,R_1),(A_1,R_1),(A_2,R_0)\},\ \{(A_0,R_1),(A_1,R_1),(A_2,R_1)\}\}.$$

From this set, we select the second element since it has the longest Euclidean distance to I_2. Consequently: $sched_W = \{(A_0, R_1), (A_1, R_1), (A_2, R_0)\}$. In the fourth iteration ($i = 3$), D_1 has again two members as displayed in Table 4, from which we select the second one with the longest distance to I_3. The solution of the algorithm is: $sched_W = \{(A_0,R_1),(A_1,R_1),(A_2,R_0),(A_3,R_1)\}$ and the final objective value is (6, 110) which dominates the constraint vector (12, 120).

8. Evaluation

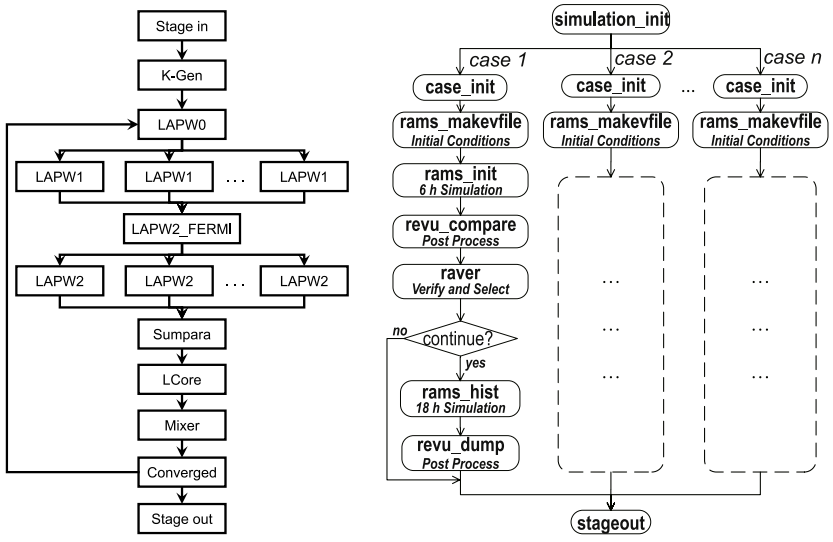
We implemented the MOLS algorithm as part of the ASKALON programming and computing environment [10] for Grid and Cloud architectures. We use in our experiments the ASKALON's backend interface to the GroudSim [15] simulator that uses trace data collected from historical execution conducted in the Austrian Grid (http://www.austriangrid.at). To estimate activity execution times on new resources, we employ the ASKALON' prediction service.

8.1. Real-world workflows

We choose two real-world applications for our experiments. First, WIEN2k [18] is a material science application for performing electronic structure calculations of solids. The WIEN2k workflow displayed in Fig. 6(a) contains two parallel sections with sequential synchronization activities in between. Second, MeteoAG [5] is an application designed for meteorological simulations based on a numerical atmospheric model. The MeteoAG workflow shown in Fig. 6(b) has a large set of simulation cases as a parallel loop, while for each simulation another nested parallel loop is executed with different parameters. Unlike WIEN2k, MeteoAG has an unbalanced structure, some parallel branches being longer than the others because of decision points in each parallel branch. Both workflows are computationally intensive applications with data transfers below 10 MB.

8.2. Simulation setup

To investigate the impact of different parameters (workflow shape, activity workload and workflow size), we ran extensive experiments on synthetic workflows alongside the real-world



(a) WIEN2k. (b) MeteoAG.

Fig. 6. Experimental real-world workflows.

Table 5
Simulation setup.

(a) Resource cost summary.

Resource	Cost ($)
Computation	0.08–1.00 (per hour)
Network (in/out)	0.01 (per GB)
Storage	0.15 (per TB per hour)

(b) Workflow size classification.

Workflow	Small	Medium	Large
WIEN2k	7–100	200–700	1000–2003
MeteoAG	71–100	5000–10 000	15 000–20 702
Random	10–500	10 000–20 000	20 000–50 000

(c) Workload classification.

Workload class	Work (A_i)
Short	100–500
Medium	500–3000
Long	3000–6000

(d) Balance classification.

Balanced	Semi-balanced	Unbalanced
$0.0 \leq \sigma^2 < 0.2$ $\wedge$ $0.0 \leq \sigma'^2 < 0.2$	$0.2 \leq \sigma^2 < 0.5$ $\wedge$ $0.2 \leq \sigma'^2 < 0.5$	$0.5 \leq \sigma^2 \leq 1.0$ $\wedge$ $0.5 \leq \sigma'^2 \leq 1.0$

ones created using a random workflow generator. Four normal distributions N, N', N'' and N''' as input parameters adjust the generated workflow characteristics (i.e. depth, balance, activity number, data transfer size), explained in the following. The normal distribution $N(\mu, \sigma^2)$ with the mean $\mu = 0.5$ and the variance $\sigma^2 \in [0.0, 1.0]$ defines the depth (the number of levels) of the workflow. A value of zero for the variance means that all workflow activities (except the entry and exit activities) are in the same level. A value one means that there is only one activity per level (sequential activities). Another normal distribution $N'(\mu', \sigma'^2)$ with the mean $\mu' = 0.5$ and the variance $\sigma'^2 \in [0.0, 1.0]$ defines how the workflow is balanced. The value of zero for the variance means a workflow is balanced and the value one represents an unbalanced workflow. Two other normal distributions N'' and N''' with arbitrary mean and variance values generate the number of activities and the size of data transfers. To simulate the heterogeneous DCI, we generated the processor speeds with a uniform distribution from 200 to 1000 MIPS with a step of 50 and considered that resources are fully connected through a 10 gigabit Ethernet network. We estimated the costs of resources based on Amazon prices, as summarized in Table 5(a). We generated failure rates using a uniform distribution in the interval [0, 0.050]. To estimate the energy consumption, we rely on the cube-root rule [4] which approximates the clock frequency of a chip as the cube-root of its power consumption. We generated the user constraints randomly within their validity ranges. To investigate the impact of the workflow size, we categorized the workflows into three classes: *small*, *medium*, and *large* based on their number of activities (see Table 5(b)). To investigate the impact of the activity workloads, we further categorized the workflows in three classes, *short*, *medium* and *large*, as displayed in Table 5(c). Finally, we study the results for *balanced*, *semi-balanced* and *unbalanced* workflows, defined based on the variance of their depth as shown in Table 5(d). Based on this experimental setup, we generated a total of 1000 real-world and 10 000 synthetic workflow experiments with the parameters uniformly distributed within the specified ranges.

8.3. Results

We analyze the experimental results in four parts: ability to satisfy user constraints, quality of schedules measured in distance to the Utopia point, and comparison with two related bi-objective algorithms (LOSS and evolutionary).

8.3.1. Constraint satisfaction

In the first step, we analyze the capability of MOLS to meet user-specified constraints in different scenarios. As shown in Fig. 7, MOLS yields better results with increasing workflow sizes. We observe that when the workflows are more balanced, the percentage of dominating solutions (i.e. the ratio between the number of solutions which dominate the target user constraints and the total number of experiments) is higher. In addition, MOLS shows better results for long activities compared to shorter ones. Fig. 8 depicts the percentage of dominating solutions for the two real workflows, showing better results for WIEN2k than MeteoAG due to its balanced structure. Nevertheless, the impact of the workflow shape on the solution decreases with larger workflows and machine sizes. Figs. 7 and 8 also indicate that for a small number of processors, MOLS often fails to find dominant solutions because of insufficient variety of resources. As soon as this number grows, the performance of the algorithm in finding dominant solutions substantially increases.

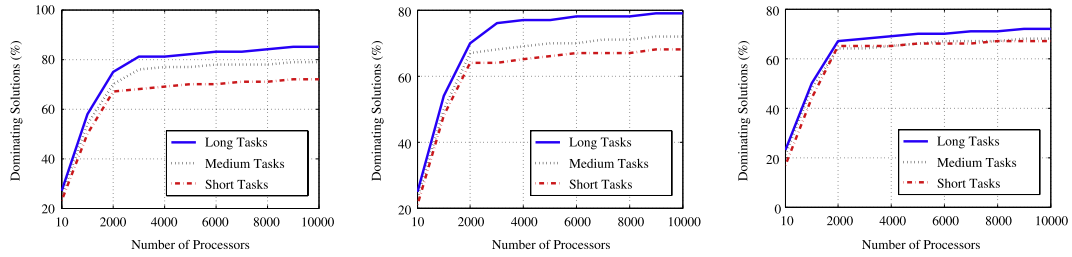
8.3.2. Distance to utopia point

In the second part of the experiments, we analyze the quality of the schedules delivered by MOLS if the users are not able to specify their constraints. In the absence of the constraint vector, the algorithm aims to find a solution as close as possible to the Utopia point. To evaluate the quality of the results, we define a new metric called *position* as the ratio of the distance between the solution and the Utopia point to the distance between the Utopia and the Nadir point:

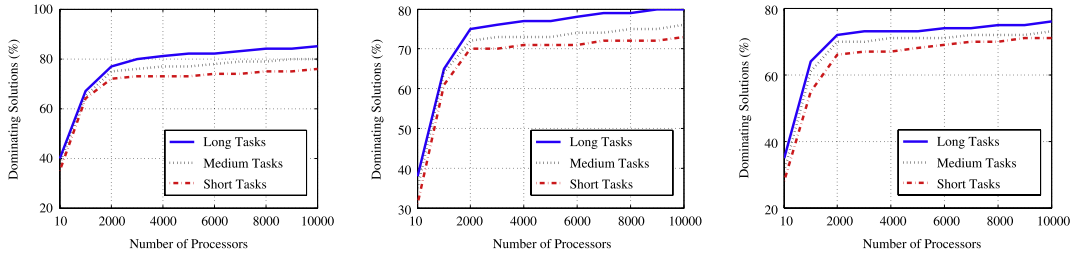
$$Position\,(sched_W) = \frac{d\,(sched_W, Utopia\ point)}{d(Nadir\ point,\ Utopia\ point)}.$$

This metric indicates how close the generated solutions are to the Utopia points (the lower, the better). Figs. 9 and 10 show that for a small number of processors, the algorithm is not able to generate solutions close to the Utopia point. For increasing processor numbers, however, the solutions continuously get closer to the Utopia point. Moreover, the results yield higher quality for increasing workflow sizes. Similar to the previous experiments, the results shown in Fig. 9 for the WIEN2k workflow are of higher quality than for MeteoAG because of its balanced structure. For increasing workflow and machine sizes, the quality of the solutions for the two workflows becomes similar. Moreover, we observe in Fig. 10 that the quality of solution for longer activities is higher.

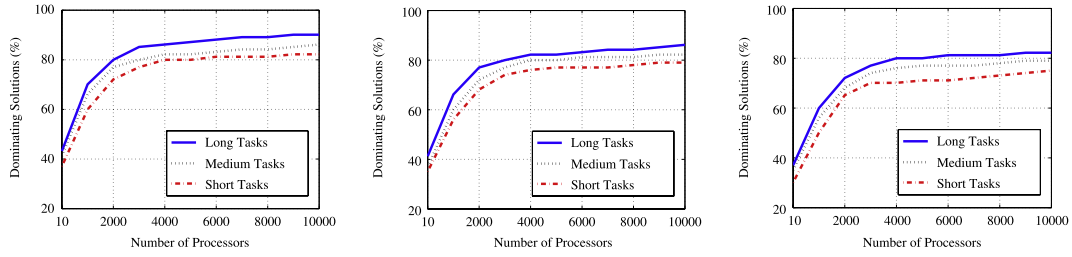
To understand the efficiency of the algorithm, it is important to investigate the dispersion of the final results compared to dispersion of the user-defined constraints. For this purpose, we conducted experiments using different workflow types, workflow sizes and number of processors. To cope with the variation of the dispersion ranges and allow direct comparison of the results, we normalized all values by scaling them to the interval [0, 100]: $\frac{data - data_{\min}}{data_{\max} - data_{\min}} \cdot 100$. We generated the constraints using a uniform random distribution shifted towards the best values by a random offset meaning that we try to create random constraints closer to the best value of each objective (this generates harder conditions for our experiments). The results are presented in Fig. 11, where the boxplots 1, 3, 5 and 7 show the dispersion of the constraints and the boxplots 2, 4, 6 and 8 show the dispersion of the solutions. We observe a remarkable improvement in the results for all four objectives, as indicated by the whiskers and the skewness towards the better values. For example, the dispersion of the solutions' makespan (boxplot 2) is around 7% for the first quartile compared to 18% for the first quartile for the constraints (boxplot 1).



(a) Small balanced. (b) Small semi-balanced. (c) Small unbalanced.

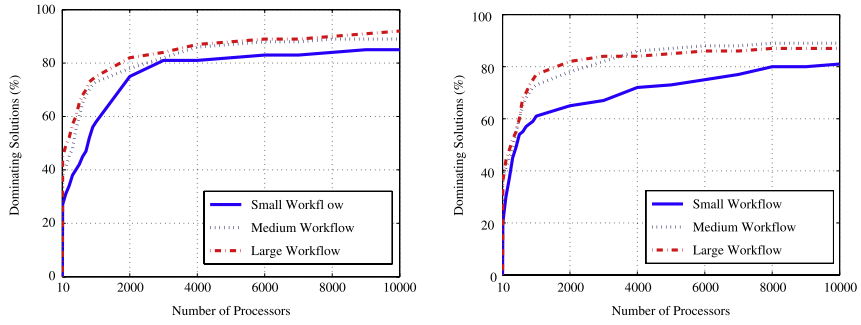


(d) Medium balanced. (e) Medium semi-balanced. (f) Medium unbalanced.



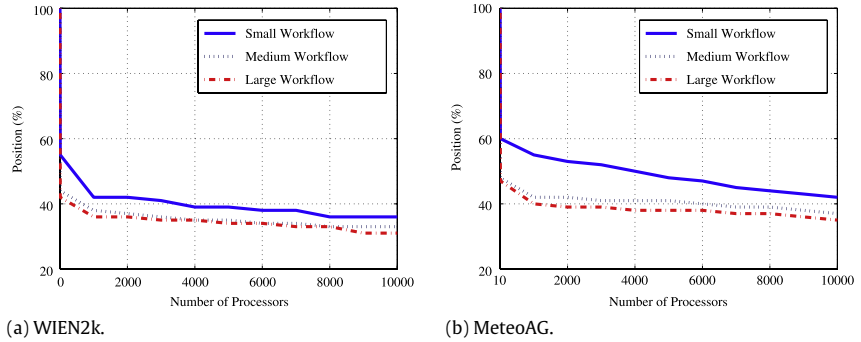
(g) Large balanced. (h) Large semi-balanced. (i) Large unbalanced.

Fig. 7. Dominating solutions for synthetic workflows.



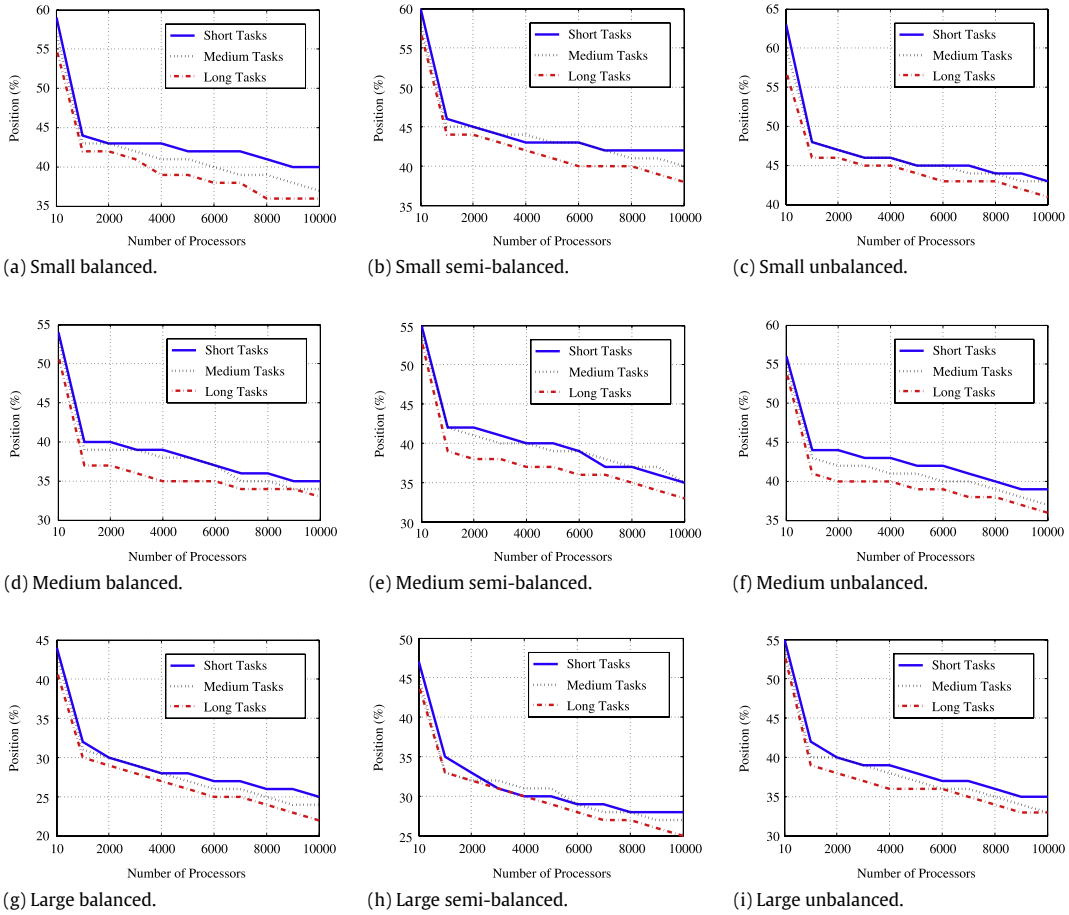
(a) WIEN2k. (b) MeteoAG.

Fig. 8. Dominating solutions for real-world applications.



(a) WIEN2k. (b) MeteoAG.

Fig. 9. Quality of solutions for real-world applications.



(a) Small balanced. (b) Small semi-balanced. (c) Small unbalanced.

(d) Medium balanced. (e) Medium semi-balanced. (f) Medium unbalanced.

(g) Large balanced. (h) Large semi-balanced. (i) Large unbalanced.

Fig. 10. Quality of solutions for synthetic workflows.

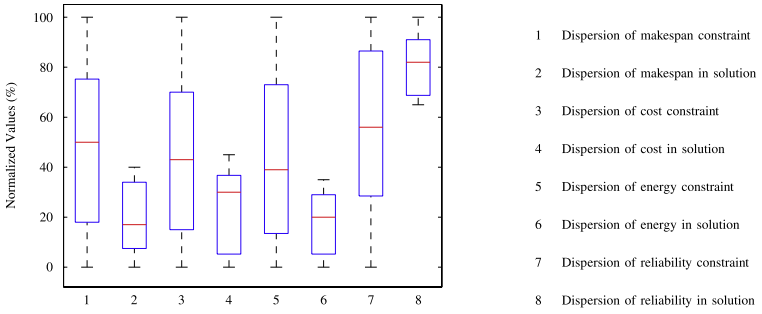
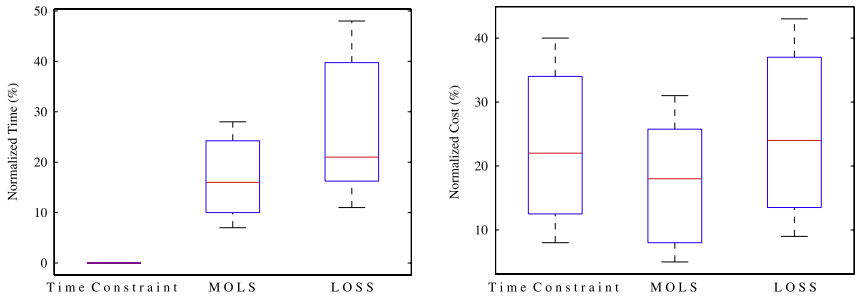


Fig. 11. Dispersion of the constraints and solutions.

8.3.3. *MOLS versus LOSS*

In the third part of the experiments, we compare MOLS with the LOSS algorithm [17] designed for optimizing for two objectives: makespan and economic cost. To make the two algorithms comparable, we assigned a weight of zero to energy and reliability in the MOLS four-objective implementation. LOSS starts with an initial mapping delivered by a makespan-driven workflow scheduling heuristic (HEFT). As long as the budget is exceeded, LOSS continuously reassigns activities to resources such that the following weight function is minimized: $LossWeight\ (A_i, R_m) = \frac{T_{new}-T_{old}}{C_{old}-C_{new}}$, where T_{old} and C_{old} are the time and cost of executing the activity A_i on the initial resource, and T_{new} and C_{new} are the time and cost of execution on the new resource R_m. If $C_{old} \leq C_{new}$, the weight function is considered zero. The algorithm tries reassigning activities by considering the smallest values of the weight function for all activities and all resources. We ran 1000 experiments using the WIEN2k and MeteoAG workflows with different workflow and machine sizes. Since LOSS only considers a constraint on budget, we considered an equal cost constraint for both algorithms and a makespan constraint equal to the minimum makespan for MOLS.

As shown in Fig. 12(a), the solutions of MOLS have in general shorter makespans compared to LOSS. The normalized time constraints in Fig. 12(a) are zero, since the makespan constraint in these experiments is the lower bound of the validity range. Fig. 12(b) proves that the LOSS algorithm delivers almost identical costs to specified cost constraint due to its approach. In contrast, the policy of MOLS is not to get as close as possible to the maximum budget, but to simultaneously decrease both cost and time by looking for dominant solutions compared to the specified constraints. A detailed comparison of MOLS and LOSS is shown in Table 6 indicating that MOLS outperforms LOSS with respect to both objectives. The reason is that MOLS does not try to find the

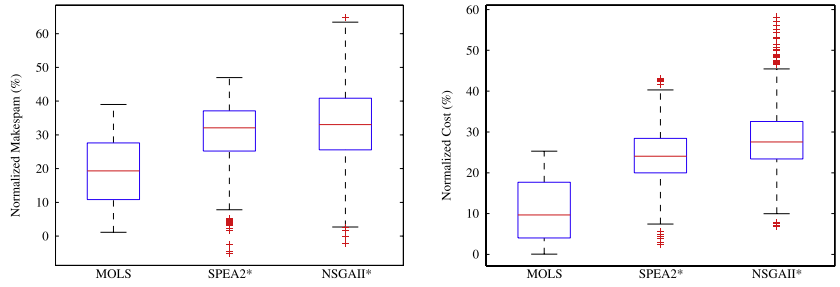


(a) Makespan dispersion. (b) Cost dispersion.

Fig. 12. Dispersion of objectives for MOLS and LOSS.

Table 6
Average objective values for MOLS and LOSS.

		MOLS			LOSS		
		Small	Medium	Large	Small	Medium	Large
WIEN2k	Time (s)	$1.8 \cdot 10^{15}$	$7.9 \cdot 10^{5}$	$1.4 \cdot 10^{6}$	$2.2 \cdot 10^{5}$	$8.3 \cdot 10^{5}$	$2.3 \cdot 10^{6}$
	Cost ($)	230	790	1250	260	850	930
MeteoAG	Time (s)	$6.4 \cdot 10^{4}$	$3.3 \cdot 10^{5}$	$3.9 \cdot 10^{6}$	$9.7 \cdot 10^{4}$	$4.0 \cdot 10^{5}$	$4.1 \cdot 10^{6}$
	Cost ($)	110	410	1060	160	500	1100



(a) Makespan dispersion. (b) Cost dispersion.

Fig. 13. Dispersion of objectives in MOLS, SPEA2* and NSGAII*.

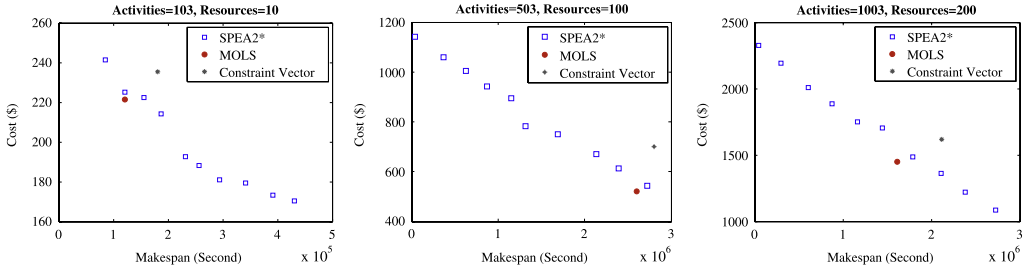


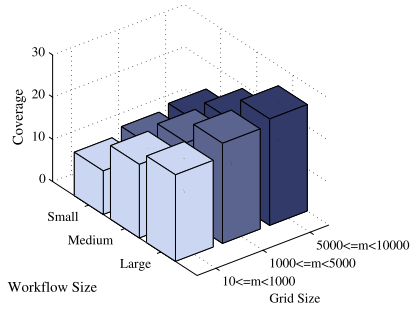
Fig. 14. Comparison of MOLS solution versus Pareto frontier of SPEA2* for three sample WIEN2k experiments.

best resource for one activity only, but searches for a resource that dominates all intermediate constraint vectors. In other words, it considers eventual deficits of previous mappings.

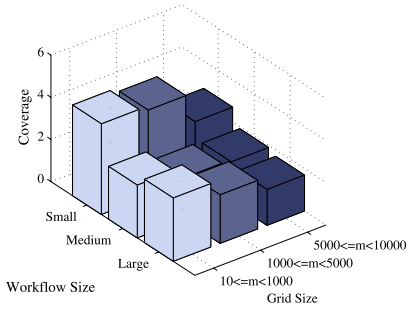
8.4. MOLS versus evolutionary algorithms

In the last part of the experiments, we compare MOLS with two multi-objective evolutionary algorithms, SPEA2 [24] and NSGA-II [6], customized in [23] for workflow scheduling under the new names SPEA2* and NSGAII*. These two algorithms are a-posteriori multi-objective optimization techniques [14] that approximate the entire Pareto set. We used jMetal library [9] to implement the evolutionary algorithms configured with the settings proposed in [23] (e.g. we set the length of the Pareto set to 10). We ran 200 experiments using the WIEN2k and MeteoAG workflows with different workflow and machine sizes and present normalized results. Fig. 13 displays the makespan and cost dispersions of MOLS, SPEA2* and NSGAII*, which indicate that the solutions of MOLS are generally better. For a better perception of the solutions found by MOLS compared to the Pareto frontiers estimated by SPEA2 and the constraint vectors, Fig. 14 represents the result of three experimental WIEN2k runs. We can observe that the solutions of MOLS dominate the constraint vector and also some of the Pareto solutions approximated by SPEA2*.

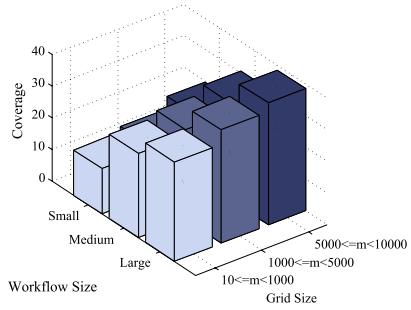
Nevertheless, in some experiments a few solutions of the evolutionary algorithms dominate the solutions of MOLS. To investigate how often this situation happens, we use the *coverage* metric [25] of a set A on a set B indicating how many members of B are dominated by the members of A. As shown in Fig. 15, the average coverage of MOLS is considerably larger than the one of



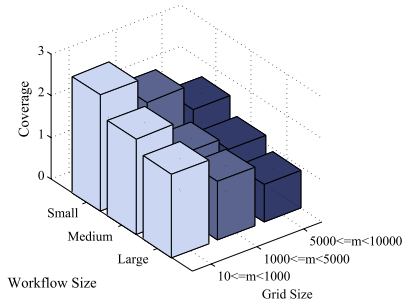
(a) Coverage of MOLS on SPEA2*.



(b) Coverage of SPEA2* on MOLS.

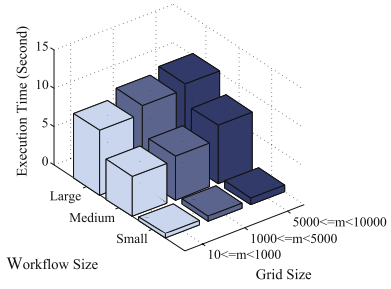


(c) Coverage of MOLS on NSGAII*.

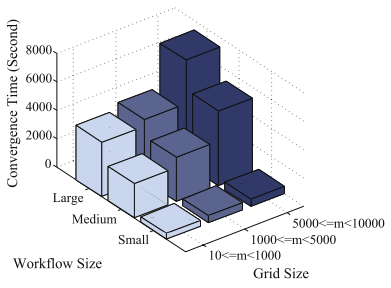


(d) Coverage of NSGAII* on MOLS.

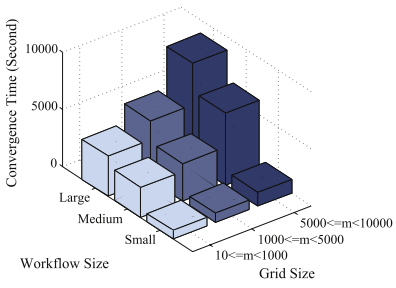
Fig. 15. Average coverage of MOLS, SPEA2* and NSGAII*.



(a) MOLS.



(b) SPEA2*.



(c) NSGAII*.

Fig. 16. Execution times of MOLS, SPEA2* and NSGAII*.

SPEA2* and NSGAII*. Consequently, although some evolutionary algorithms particularly SPEA2 are efficient algorithms for many problems [24], these experiments prove that MOLS is more suitable than SPEA2* and NSGAII* for workflow scheduling. The main reason for the superiority of MOLS is that it considers the activity precedence constraints in all phases of the algorithm, in contrast to the evolutionary algorithms that slowly converge to a local minimum. Fig. 16 shows that evolutionary algorithm needs significantly longer execution times to converge to solutions of a similar quality with the ones of MOLS.

9. Conclusion

In this paper, we proposed a generic multi-objective scheduling heuristic for workflow applications in heterogeneous DCIs. We tailored this generic model for a four-objective case study comprising makespan, economic cost, energy consumption and reliability. Considering user-specified constraint for objectives, the algorithm follows a dual strategy based on Pareto dominance relationships: maximize the distance to the constraint vector for dominant solutions and minimize it otherwise. We conducted extensive evaluation for real-world and synthetic workflow applications and observed that, in most experiments, our solutions dominate the user-specified constraints. Compared to the Pareto set estimated by SPEA2* and NSGAII*, our algorithm is considerably faster and delivers better solutions. Extending the MOLS algorithm for dynamic scenarios considering model inaccuracies for both application and DCI is an interesting future direction for this work.

Acknowledgments

University of Innsbruck (Doktoratsstipendium NEU aus der Nachwuchsförderung), Austrian Science Fund (project TRP 237-N23), and Standortagentur Tirol (project RainCloud) funded this research.

References

[1] I. Assayad, A. Girault, H. Kalla, A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints, in: 2004 International Conference on Dependable Systems and Networks, IEEE Computer Society, 2004, pp. 347–356.
[2] A. Benoit, M. Hakem, Y. Robert, Multi-criteria scheduling of precedence task graphs on heterogeneous platforms, Comput. J. 53 (6) (2010) 772–785.
[3] A. Benoit, H. Kosch, V. Rehn-Sonigo, Y. Robert, Multi-criteria scheduling of pipeline workflows (and application to the jpeg encoder), Int. J. High Perform. Comput. Appl. 23 (2) (2009) 171–187.

[4] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, P.W. Cook, Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors, IEEE Micro 20 (6) (2000) 26–44.
[5] W.R. Cotton, R.A. Pielke, R.L. Walko, G.E. Liston, C.J. Tremback, H. Jiang, R.L. McAnelly, J.Y. Harrington, M.E. Nicholls, G.G. Carrio, J.P. McFadden, RAMS 2001: current status and future directions, Meteorol. Atmos. Phys. 82 (2003) 5–29.
[6] K. Deb, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[7] A. Doğan, F. Özgüner, Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems, Comput. J. 48 (3) (2005) 300–314.
[8] A. Doğan, F. Özgüner, Scheduling of a meta-task with QoS requirements in heterogeneous computing systems, J. Parallel Distrib. Comput. 66 (2) (2006) 181–196.
[9] J.J. Durillo, A.J. Nebro, jMetal: a Java framework for multi-objective optimization, Adv. Eng. Softw. 42 (10) (2011) 760–771.
[10] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, J.Q. Stefan Podlipnig, M. Siddiqui, H.-L. Truong, A. Villazón, M. Wieczorek, ASKALON: a development and grid computing environment for scientific workflows, in: I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, 2007, pp. 450–471.
[11] A.K.M. Khaled, A. Talukder, M. Kirley, R. Buyya, Multiobjective differential evolution for scheduling workflow applications on global grids, Concurr. Comput.: Pract. Exp. 21 (13) (2009) 1742–1756.
[12] S.U. Khan, C. Ardil, Energy efficient resource allocation in distributed computing systems, World Acad. Sci. Eng. Technol. 56 (2009) 667–673.
[13] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Multicriteria aspects of grid resource management, in: Grid Resource Management, in: International Series in Operations Research & Management Science, vol. 64, Springer, 2004, pp. 271–293.
[14] R. Marler, J. Arora, Survey of multi-objective optimization methods for engineering, Struct. Multidiscip. Optim. 26 (6) (2004) 369–395.
[15] S. Ostermann, K. Plankensteiner, R. Prodan, Using a new event-based simulation framework for investigating resource provisioning in clouds, Sci. Program. 19 (2–3) (2011) 161–178.
[16] J.-F. Pineau, Y. Robert, F. Vivien, Energy-aware scheduling of flow applications on master–worker platforms, in: Euro-Par 2009 Parallel Processing, in: LNCS, vol. 5704, 2009, pp. 281–292.
[17] R. Sakellariou, H. Zhao, E. Tsiakkouri, M.D. Dikaiakos, Scheduling workflows with budget constraints, in: S. Gorlatch, M. Danelutto (Eds.), Integrated Research in GRID Computing, Springer, 2007, pp. 189–202.
[18] K. Schwarz, P. Blaha, G.K.H. Madsen, Electronic structure calculations of solids using the WIEN2k package for material sciences, Comput. Phys. Comm. 147 (1–2) (2002) 71–76.
[19] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (2002) 260–274.
[20] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the grid, Future Gener. Comput. Syst. 25 (3) (2009) 237–256.
[21] J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: 1st Workshop on Workflows in Support of Large-Scale Science, IEEE Computer Society, 2006, pp. 1–10.
[22] J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: First International Conference on e-Science and Grid Computing, IEEE Computer Society, 2005, p. 8.
[23] J. Yu, M. Kirley, R. Buyya, Multi-objective planning for workflow execution on grids, in: 8th International Conference on Grid Computing, IEEE Computer Society, 2007, pp. 10–17.
[24] E. Zitzler, M. Laumanns, L. Thiele, Spea2: improving the strength Pareto evolutionary algorithm, in: Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems, International Center for Numerical Methods in Engineering, 2002, pp. 95–100.
[25] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.

Hamid Mohammadi Fard received his Master degrees in Computer Science in Iran and between 1998 and 2009 he worked in Iran's telecommunication industry. Since 2009 he has been working as a researcher and Ph.D. student at the Institute of Computer Science, University of Innsbruck, Austria.

Radu Prodan received the Ph.D. degree from the Vienna University of Technology, Austria, in 2004. He is currently an Associate Professor at the Institute of Computer Science, University of Innsbruck, Austria. His research in the area of parallel and distributed systems comprises programming methods, compiler technology, performance analysis and scheduling. He is the author of more than 130 journal and conference papers and one book. He was the recipient of an IEEE best paper award.

Thomas Fahringer is a Full Professor of Computer Science at the Institute of Computer Science, University of Innsbruck. His research interests are programming languages, performance tools, debuggers, compiler analysis and program optimization for parallel and distributed systems. He participated in numerous EU-funded, international and national projects. He is the author of more than 200 papers including 30 journal articles and four books. Fahringer was the recipient of three IEEE/ACM best paper awards.