2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments

Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo and Thomas Fahringer
Institute of Computer Science, University of Innsbruck, Austria
{hamid, radu, juan, tf}@dps.uibk.ac.at

***Abstract*—Traditional scheduling research usually targets makespan as the only optimization goal, while several isolated efforts addressed the problem by considering at most two objectives. In this paper we propose a general framework and heuristic algorithm for multi-objective static scheduling of scientific workflows in heterogeneous computing environments. The algorithm uses constraints specified by the user for each objective and approximates the optimal solution by applying a double strategy: maximizing the distance to the constraint vector for dominant solutions and minimizing it otherwise. 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. We implemented the algorithm as part of the ASKALON environment for Grid and Cloud computing. Results for two real-world applications demonstrate that the solutions generated by our algorithm are superior to user-defined constraints most of the time. Moreover, the algorithm outperforms a related bi-criteria heuristic and a bi-criteria genetic algorithm.**

Keywords-heterogeneous computing systems; workflow scheduling; multi-objective optimization; Grids and Clouds

I. INTRODUCTION

Scheduling a set of different tasks in heterogeneous environments is one of the traditional challenges in parallel and distributed computing. Scientific workflows have emerged in the last decade as an attractive paradigm for programming large-scale applications in heterogeneous computing environments such as computational Grids. Depending on the objective function such as the overall makespan, the scheduling problem can be NP-complete requiring heuristics for approximating optimal solutions. Typically, there are different actors in decentralized heterogeneous environments, each of them viewing the problem from a different perspective. Scientists are usually interested in minimizing the execution time of their application, whereas system administrators are often interested in maximizing the resource utilization, job throughput, or user fairness. Real-world scheduling problems are therefore frequently confronted with a multi-objective scenario in which many of these important objectives are conflicting. For example, powerful processors are typically rented by Cloud computing providers at a higher price, consume more energy, and may become unreliable due to the large number of requests and user contention. Today, related scheduling work is typically restricted to optimizing one objective (usually makespan), while some isolated approaches tried to optimize across two criteria [1, 2, 3]. A generic scheduling framework and heuristic-based algorithms for optimizing multiple conflicting criteria are still missing.

In this paper, we propose a new polynomial multi-objective scheduling algorithm for scientific workflow applications in heterogeneous environments such as computational Grids. The user specifies a vector comprising a constraint for each individual objective, which is used as an approximation of the position of the final solution in the solution space. The algorithm approximates an optimal solution using a list scheduling heuristic combined with multi-objective optimization theory targeting two goals: maximizing the distance to the constraint vector for dominant solutions and minimizing it otherwise.

The paper is organized as follows. Section II introduces several key concepts of the multi-objective optimization theory. In Section III, we describe the generic application, objective, and platform models underneath our approach, which we instantiate in Section IV by a four-objective case study. Section V presents our generic multi-objective list scheduling heuristic, followed by an evaluation using a broad set of experiments scenarios in Section VI. Section VII summarizes the related work. Section VIII concludes the paper.

II. BACKGROUND

In this section, we introduce several important concepts of *multi-objective optimization* theory [4] used in our proposed algorithm involving two steps: finding a set of optimal solutions and selecting the fittest solutions according to user preferences. As part of multi-objective optimization problems, two spaces must be distinguished: 1) *solution* (or *design*) *space* X comprising all feasible solutions, for example the complete set of possible schedules of an application; and 2) o*bjective* (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.

In the following, we outline a few critical concepts of multi-objective optimization theory required to understand:

- *Pareto dominance:* 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;
- *Pareto optimality:* 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;
- *Pareto-optimal set:* The set of all Pareto-optimal solutions is called Pareto-optimal set;
- *Pareto frontier:* The image set of all members of a Pareto-optimal set in the objective space is called Pareto frontier;
- *Utopia point* (*ideal point*): A vector comprising the best possible values for all objectives is called Utopia point. Such a vector typically dominates the Pareto-optimal set and is therefore impossible to realistically achieve;

978-0-7695-4691-9/12 $26.00 © 2012 IEEE
DOI 10.1109/CCGrid.2012.114

- *Nadir point:* A vector comprising the worst possible values for all existing objectives is called Nadir point.

III. Model

In this section, we first formally define the abstract application, multi-objective, and platform models underneath our approach. Afterwards, we outline the implementation of our proposed algorithm as part of the ASKALON [5] programming and computing environment for Grid and Cloud infrastructures.

A. Application Model

We model a *workflow application* as a directed acyclic graph (DAG) $W = (A, D)$ consisting of a set of n *activities* $A = \bigcup_{i=1}^{n}\{A_i\}$ interconnected through a set of control flow and data flow dependencies: $D = \{(A_i, A_j, Data_{ij}) | (A_i, A_j) \in A \times A\}$, where $Data_{ij}$ represents the size of the data which needs to be transferred to A_j from A_i before its execution. We use $pred(A_j)$ to denote the *predecessor* set of activity A_j, where $A_i \in pred(A_j)$. Finally, we assume the availability of the computational *workload* of each activity A_i, by the number of instructions per second, denoted as $work(A_i)$.

B. Multi-Objective Model

Let $(O_1, \dots, O_k)$ denote the *objective vector* expressing a set of k objectives to be simultaneously optimized. Moreover, we assume the availability of a *constraint vector* $(C_1, \dots, C_k)$ provided by a user that defines the constraint values for the k objectives, considered as *soft constraints* that shall be fulfilled in a best-effort fashion (in contrast to hard constraints which must be handled by all circumstances [6]). 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 IV, 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* $(w_1, \dots, w_k)$ such that $\sum_{i \in [k]} w_i = 1$, which defines the contribution of each objective in the decision making process. The largest weight value has the most significant impact on the final solution. If the actors are 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.

C. 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* $(v_{1j}, \dots, v_{kj})$, which we will define in Section IV for our four-objective case study. The resources are assumed to be connected in a full mesh network, where each point-to-point network connection has a different bandwidth. We also assume that all resources are non-preemptive (i.e. it is not allowed to suspend an activity and resume it later on). Furthermore, we assume the possibility of reserving resources in advance. Finally, we define a *workflow schedule* as a mapping $sched: A \to R$ 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 $|X| = m^n$.

D. Execution Model

We implemented the proposed algorithm as part of the ASKALON environment which supports the development and execution of scientific workflows on Grid and Cloud infrastructures. As shown in Fig. 1, 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 submit them as jobs. 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, ASKALON is also interfaced at the backend with the GroudSim simulator [7] that supports modeling of Grid and Cloud computational and network resources, including job submissions, file transfers, failure models, background load, and cost models.

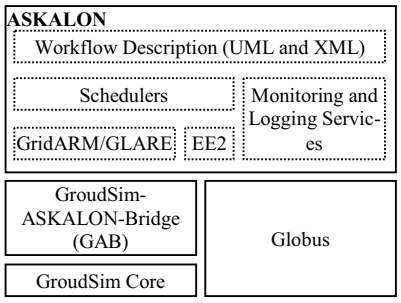


Figure 1. Simplified architecture of the ASKALON environment.

IV. Four-Objective Case Study

Based on the generic model defined in Section III.A, 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 with respect to several important classification criteria [8], summarized in Table 1.

Table 1. Objective Function Classification Based on [8]

Objective	*Structural dependency*	*Aggregation*	*Complexity**	*Direction*
Makespan	Structure-dependent	Additive	$\mathcal{O}(m^n)$	Min
Economic cost	Structure-independent	Additive	$\mathcal{O}(m \cdot n)$	Min
Energy	Structure-independent	Additive	$\mathcal{O}(m \cdot n)$	Min
Reliability	Structure-independent	Multiplicative	$\mathcal{O}(m \cdot n)$	Max

* m and n represent the number of tasks, respectively resources.

A. Maskespan

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

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

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

$$end(A_i, R_j) = \begin{cases} O_1(A_i, R_j), & \text{pred}(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \{end(A_p, sched(A_p)) + O_1(A_i, R_j)\}, & \text{pred}(A_i) \neq \emptyset. \end{cases}$$

Consequently, the workflow makespan is given by the longest completion time of its activities:

$$O_1(W) = \max_{i \in [1..n]} \{end(A_i, sched(A_i))\}.$$

To estimate a feasible valid interval $C_1 \in [O_1^{(\min)}, O_1^{(\max)}]$, we use the HEFT algorithm [9] to compute the lower bound $O_1^{(\min)}$ (an NP-complete problem), and the sequential execution of the activities on the slowest computing machine to compute the upper bound $O_1^{(\max)}$.

B. 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:

$$O_2(A_i, R_j) = O_1(A_i, R_j) \cdot v_{2j}^{(1)} + TotalData(A_i) \cdot O_1(A_i, R_j) \cdot v_{2j}^{(2)} + Input'(A_i) \cdot v_{2j}^{(3)} + Output'(A_i) \cdot v_{2j}^{(4)},$$

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:

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

The validity range $C_2 \in [O_2^{(\min)}, O_2^{(\max)}]$ 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.

C. Energy Consumption

In terms of *energy consumption*, our focus is on computational-intensive applications. Therefore, we only consider the computation part of activities as significantly contributing to the energy consumption, while the data transfers and storage time are ignored. 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}, which we model according to [10, 11]:

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

The workflow energy consumption is also an additive structure-independent criterion that can be calculated as follows:

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

Energy consumption can be optimally solved by mapping all activates 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 $C_3 \in [O_3^{(\min)}, O_3^{(\max)}]$, where $O_3^{(\max)}$ is again calculated by scheduling each activity on the most energy consuming resource.

D. 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 using an exponential distribution [3, 12]:

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

In contrast to makespan, cost, and energy, reliability is a multiplicative objective [3, 12]. Consequently:

$$O_4(W) = \prod_{i=1}^{n} e^{-v_{4j} \cdot O_1(A_i, R_j)}, \text{ where } sched(A_i) = R_j.$$

In terms of the validity range $C_4 \in [O_4^{(\min)}, O_4^{(\max)}]$, 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 three objectives.

V. Multi-Objective List Scheduling for a Workflow

We propose a *multi-objective list scheduling* (MOLS) algorithm that attempts to find a solution which dominates the constraint vector or otherwise a solution which converges to the constraint vector. The MOLS algorithm is based on three major phases, outlined in pseudo-code in Algorithm 1. In the first phase (lines 5—11), 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 12), it assigns a rank to each activity of the workflow and sorts them in ascending order. Finally in the third phase (lines 14—28), the algorithm attempts to allocate the most appropriate resource to each activity with respect to the estimated sub-constraints.

A. Phase One: Constraint Vector Partitioning

In the first phase, we compute the validity ranges of each objective O_i (line 6) and select a constraint C_i for each O_i (line 7). Afterwards, we partition the constraint vector $(C_1, \dots, C_k)$ among the workflow activities to create a *partial constraint vector* $\left(C_1^{(i)}, C_2^{(i)}, \dots, C_k^{(i)}\right)$ for each activity A_i (lines 9—11) representing an estimation of the constraints for each individual activity. 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. In the following, we describe 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.

Algorithm 1: MOLS algorithm

1: **Input:** *W = (A, D): workflow application,* $A = \bigcup_{i=1}^{n}\{A_i\}$**;**
2: $(O_1, \dots, O_k)$ and $(w_1, \dots, w_k)$*: objective and weight vectors*;
3: $R = \bigcup_{j=1}^{m}\{R_j\}$*: resource set*;
4: **Output:** $\bigcup_{i=1}^{n}\{(A_i, sched(A_i))\}$*: workflow schedule;*
5: **for** *i* ← 1 **to** *k* // ***Phase One***
6: *Compute validity ranges:* $\left[O_i^{(\min)}, O_i^{(\max)}\right]$;
7: *Select constraint:* $C_i \in \left[O_i^{(\min)}, O_i^{(\max)}\right]$;
8: **end for**
9: **for** *i* ← 1 **to** *n*
10: *Compute partial constraint vector:* $\left(C_1^{(i)}, \dots, C_k^{(i)}\right)$;
11: **end for**
12: *B* ← *sort*(*A*, *B-level*) ; // ***Phase Two***
13: *Sol* ← ∅;
14: **for** *i* ← 1 **to** *n* **do** // ***Phase Three***
15: *Compute intermediate constraint vector* I_i;
16: *Generate all solutions:* $M \leftarrow \bigcup_{j=1}^{m}\left\{\left(Sol \cup (B_i, R_j)\right)\right\}$;
17: $D_1 = \{S \in M \mid O(S) \succ I_i\}$;
18: **if** $D_1 \neq \emptyset$ **then**
19: $Sol \leftarrow S \in D_1 \mid \forall\, S' \in D_1, d(O(S), I_i) \geq d(O(S'), I_i)$, *where:*

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

20: **else**
21: $T_i \leftarrow S \in M \mid \forall\, S' \in M, d(O(S), I_i) \leq d(O(S'), I_i)$;
22: $D_2 \leftarrow \{S \in M \mid O(S) \succ T_i\}$;
23: **if** $D_2 \neq \emptyset$ **then**
24: $Sol \leftarrow S \in D_2 \mid \forall\, S' \in D_2, d(O(S), T_i) \geq d(O(S'), T_i)$;
25: **else** $Sol \leftarrow T_i$;
26: **endif**
27: **endif**
28: **endfor**
29: **return** *Sol*;

First, the makespan (or time) 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 individual activities. To accurately partition a time constraint, Algorithm 2 creates first in line 4 the set of all paths from the first to the last activity. In line 6, we assign the longest path in this set to the variable CP, where the length is calculated by summing up the workloads of all activities in the path. The CP is then removed from the $pathSet$ in line 7. Afterwards, we build two sets: $assigned$ is the set of activities in CP whose time constraints are already calculated (line 8) and $unassigned$ is the rest (line 9). In line 10 we calculate the sum of the time constraints of the $assigned$ activities and in line 11 the workload sum of the $unassigned$ ones. We use these values in line 13 to calculate the time constraints of all the $unassigned$ tasks in CP and repeat the process for all the paths in $pathSet$. We trace this algorithm using an example in Section V.E for a better understanding.

Algorithm 2: Time constraint partitioning

1: **Input:** *W = (A, D): workflow application,* $A = \bigcup_{i=1}^{n}\{A_i\}$;
2: C_1*: workflow time constraint*;
3: **Output:** $C_1^{(i)}$*: time constraints of activities in* $A_i, \forall i \in [n]$*;*
4: $pathSet$ ← set of all paths from the first to the last activity;
5: **while** ($pathSet \neq \emptyset$)
6: $CP \leftarrow$ the longest path in $pathSet$;
7: Remove($pathSet$, CP);
8: $assigned \leftarrow \bigcup\left\{A_i \mid \left(A_i \in CP \wedge C_1^{(i)} \text{ is calculated}\right)\right\}$;
9: $unassigned \leftarrow \bigcup\{A_i \mid (A_i \in CP \wedge A_i \notin assigned)\}$;
10: $spentTime \leftarrow \sum_{A_i \in assigned} C_1^{(i)}$;
11: $unassignedLoad \leftarrow \sum_{A_i \in unassigned} work(A_i)$;
12: **for** all $A_i \in unassigned$
13: $C_1^{(i)} \leftarrow \frac{(C_1 - spentTime)}{unassignedLoad} \cdot work(A_i)$;
14: **endfor**
15: **endwhile**

Second, the contribution of the economic cost constraint of each activity is proportional to its cost compared to the cost of the whole workflow averaged over all resources:

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

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 can be partitioned as follows:

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

Finally, we partition the reliability constraint using two abstract parameters: the average Grid failure rate λ_{Grid} and the average Grid speed v_{Grid}. Thus, we estimate the partial reliability constraint of activity A_i as follows:

$$C_4^{(i)} = e^{-\lambda_{Grid} \frac{work(A_i)}{v_{Grid}}}.$$

Since reliability is a multiplicative objective:

$$C_4 = \prod_{i=1}^{n} C_4^{(i)} = \prod_{i=1}^{n} e^{-\lambda_{Grid} \frac{work(A_i)}{v_{Grid}}} = e^{-\frac{\lambda_{Grid}}{v_{Grid}} \sum_{i=1}^{n} work(A_i)}.$$

After applying the natural logarithm function to both sides of this equation, the ratio of the abstract parameters is:

$$\frac{\lambda_{Grid}}{v_{Grid}} = -\frac{\ln(C_4)}{\sum_{i=1}^{n} work(A_i)}.$$

By substituting this ratio in the $C_4^{(i)}$ definition, we obtain:

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

We give an example in Section V.E.

B. Phase Two: Activity Ordering

In the second phase of the algorithm, the workflow activities are ranked in an ascending list B (line 12), which determines the scheduling order of activities. List B is therefore a new representation of list A that comprises the same activities but in a new order. In our four-objective case study, we are using for ranking the *bottom level* (*B-level)* of each activity, defined in graph theory as the longest path to the last activity including the activity itself. As discussed in our previous work [14], the reason for using the makespan for ordering the workflow activities is twofold: it is the only structure-dependent objective that preserves the precedence relationships in the ordered list and it is the objective that has the largest impact on the other three objectives.

C. Phase Three: Activity Mapping

In the mapping phase, the activities from the ordered list B are traversed and scheduled onto the "best" resource according to the k objectives. At each iteration $i \in [1..n]$ (lines 14—28), we add a new mapping $(B_i, sched(B_i))$ to an *intermediate solution* $Sol_i = \cup_{j=1}^{i}(B_j, sched(B_j))$. The final intermediate solution Sol_n is the output of the algorithm (line 29).

Let $I_i = (I_{1i}, \dots, I_{ki})$ denote the *intermediate constraint vector* at mapping step i (line 15) representing a sub-constraint vector for the sub-workflow comprising the activities $\{B_j \mid j \in 1..i\}$. In the four-objective case study, I_i is calculated by using the activities' partial constraint vectors, calculated as defined in Section IV for each objective O_i. The m possible solutions in every iteration i are first stored in the set M (line 16), where m is the number of resources:

$$M = \cup_{j=1}^{m}(Sol_{i-1} \cup (B_i, R_j)).$$

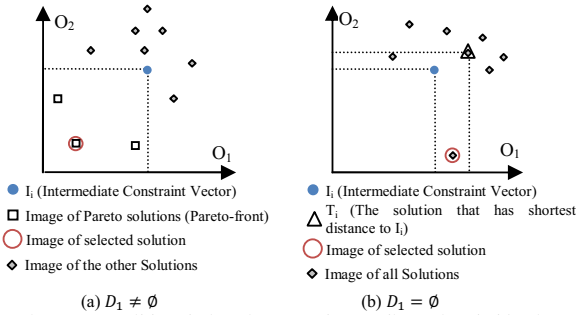
To find the "best" resource in each iteration i, all solutions in M whose objectives dominate the intermediate constraint vector I_i are added to the set D_1 (line 17): $D_1 = \{S \in M \mid O(S) \succ I_i\}$, where $O(S)$ represents the values of the objectives of the solution S. If $D_1 \neq \emptyset$, (cf. Fig. 2a), we select the member of this set with the longest weighted *Euclidian distance* $d(O(S), I_i)$, since we want to improve our solution on all objectives simultaneously (lines 18—19). One should notice that this solution is Pareto optimal, otherwise there would exist another point with a longer distance:

$$Sol_i = S \in D_1 \mid \forall S' \in D_1, d(O(S), I_i) \geq d(O(S'), I_i),$$

where $d(O(S), I_i) = \sqrt{\sum_{j=1}^{k} w_j \cdot [N(O_j(S)) - N(I_{ji})]^2}$. Since different objectives are usually incomparable, we normalize them by using the following *normalization function* [4]:

$$N: [O^{(\min)}, O^{(\max)}] \rightarrow [0,1], N(x) = \frac{x - O^{(\min)}}{O^{(\max)} - O^{(\min)}}.$$

Therefore, $N(O_j(S))$ and $N(I_{ji})$ represent the j^{th} normalized objective values of S and I_i, respectively. If $D_1 = \emptyset$ (cf. Fig. 2b), we first select a solution called T_i with the shortest weighted distance to I_i (line 21): $T_i = S \in M \mid \forall S' \in M, d(O(S), I_i) \leq d(O(S'), I_i)$, and then generate the set D_2 comprising the solutions in M that dominate T_i (line 22), similar to the previous case. If $D_2 \neq \emptyset$, we choose the solution with the longest distance (according to the values of objectives) to T_i (lines 23—24) as the final solution. If $D_2 = \emptyset$, we choose T_i as the final solution (line 25).



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

Figure 2. Possible solutions in a sample two-dimensional objective space.

D. Time complexity analysis

In this section, we theoretically analyze the time complexity of the MOLS algorithm.

Theorem 1. The time complexity of the four-objective MOLS case study for a workflow with n activities and a heterogeneous system with m resources is $O(m \cdot n^3)$.

Proof. In the first phase of the algorithm (lines 5—8), the most time consuming step is computing the makespan's lower bound. In case of the HEFT algorithm, this part is of the complexity $O(m \cdot n^2)$ [9]. In computing the partial constraint vectors (lines 9—11), the most complex part is again related to the makespan structure-dependent objective which consumes $O(n + e)$ 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 $O(n^2)$. Therefore, the time complexity of the MOLS's first phase is $O(m \cdot n^2)$. In the second phase (line 12), the B-level computation has a complexity of $O(n + e)$ or $O(n^2)$ in the worst case. Since sorting the list B of activities also has a time complexity of $O(n^2)$ in the worst case, the time complexity of the second phase is $O(n^2)$. In the third phase (lines 14—28), the time complexity of the i^{th} loop iteration is $O(m \cdot i^2)$ because of the intermediate constraint vector I_i computation in line 15. 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 $O(m \cdot n^3)$. Since the time complexity of the third phase is dominant, the complexity of the algorithm is $O(m \cdot n^3)$. □

As described before, the makespan has the major impact from the four objectives on the time complexity of the algorithm. Removing the other three objectives brings no considerable reduction on the time complexity, however, in the absence of makespan the complexity is decreased to $O(m \cdot n^2)$.

E. Illustrative Example

In this section, we present an example that illustrates the individual steps of the MOLS algorithm. Suppose we have the workflow depicted in Fig. 3 and the resource environment summarized in Table 2. For simplicity of the calculations, we consider only two objectives: makespan and cost with equal weights of one and no data transfer between activities. First of all, we calculate the validity range of each objective, as described in Section IV.A. The lower bound of makespan is the solution delivered by the HEFT algorithm 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 is given by mapping each activity onto the most expensive R_1 resource: $O_2^{(max)} = (5+4+2+3).\frac{10}{1} = 14$. Thus, $C_2 \in [105, 140]$. From the calculated validity ranges, we assume that the user specifies the constraint vector: $(C_1, C_2) = (12, 120)$.

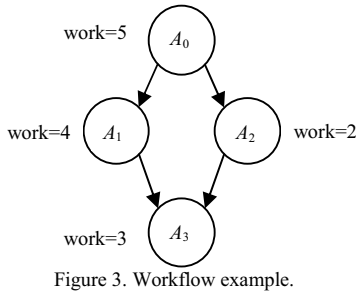


Figure 3. Workflow example.

TABLE 2. RESOURCE ENVIRONMENT EXAMPLE.

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

In the *first phase*, the algorithm calculates the partial constraint vectors. For time constraint partitioning, we use the 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 3. Cost constraints are partitioned simpler, as defined in Section V.A, 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, respectively the fourth column of Table 4.

TABLE 3. ITERATION STEPS OF ALGORITHM 2.

Line	*First Iteration*	*Second Iteration*
6	$CP = (A_0, A_1, A_3)$	$CP = (A_0, A_2, A_3)$
7	$assigned = \Phi$	$assigned = \{A_0, A_1, A_3\}$
8	$unassigned = \{A_0, A_1, A_3\}$	$unassigned = \{A_2\}$
9	$spentTime = 0$	$spentTime = 3 + 5 = 8$
10	$unassignedLoad = 12$	$unassignedLoad = 2$
12	$C_1^{(0)} = \frac{12-0}{12} \cdot 5 = 5$	$C_1^{(2)} = \frac{12-8}{2} \cdot 2 = 4$
12	$C_1^{(1)} = \frac{12-0}{12} \cdot 4 = 4$	—
12	$C_1^{(3)} = \frac{12-0}{12} \cdot 3 = 3$	—

TABLE 4. PARTIAL CONSTRAINTS AND B-LEVELS FOR THE EXAMPLE WORKFLOW.

i	*Activity*	$C_1^{(i)}$	$C_2^{(i)}$	*Partial Constraint Vector*	*B-level*
0	A_0	5	600/14 ≅ 43	(5, 43)	3 + 4 + 5 = 12
1	A_1	4	480/14 ≅ 34	(4, 34)	3 + 4 = 7
2	A_2	4	240/14 ≅ 17	(4, 17)	3 + 2 = 5
3	A_3	3	360/14 ≅ 26	(3, 26)	3

In the *second phase*, we sort the activities in list B (based on their B-level) as shown in the last column of Table 4, which yields the sequence list $[A_0, A_1, A_2, A_3]$.

In the *third phase*, we iterate over the sorted activity list and find an appropriate mapping of each activity in every iteration (see Table 5). 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, $Sol_0 = \{(A_0, R_1)\}$. The second iteration of the algorithm operates similarly, as summarized in Table 5 in the row $i = 1$. In the third iteration ($i = 2$), we create the set D_1 of solutions in $M = \begin{Bmatrix}\{(A_0, R_1), (A_1, R_1), (A_2, R_0)\} \\ \{(A_0, R_1), (A_1, R_1), (A_2, R_1)\}\end{Bmatrix}$ 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 $D_1 = \begin{Bmatrix}\{(A_0, R_1), (A_1, R_1), (A_2, R_0)\}, \\ \{(A_0, R_1), (A_1, R_1), (A_2, R_1)\}\end{Bmatrix}$. From this set, we select the second element since it has the longest Euclidian distance to I_2. Consequently: $Sol_2 = \{(A_0, R_1), (A_1, R_1), (A_2, R_0)\}$. In the fourth iteration ($i = 3$), D_1 has again two members as displayed in Table 5, from which we select the second member with the longest distance to I_3. The final solution of the algorithm is: $Sol_3 = \{(A_0, R_1), (A_1, R_1), (A_2, R_0), (A_3, R_1)\}$ and the final objective value is $(6, 110)$ which dominates the constraint vector $(12, 120)$.

TABLE 5. ITERATION STEPS OF THE MAPPING PHASE

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

VI. Evaluation

We implemented the MOLS algorithm as part of the ASKALON programming and computing environment for Grid and Cloud architectures. We use in our experiments the ASKALON's backend interface to the GroudSim 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 [15].

A. Experimental Setup

We choose two real-world applications for our experiments. WIEN2k [16] is a material science workflow for performing electronic structure calculations of solids. MeteoAG [17] is a workflow designed for meteorological simulations based on numerical atmospheric model. The WIEN2k workflow contains two parallel sections with sequential synchronization activities in between. The MeteoAG workflow has a large set of simulation cases as a parallel loop; for each simulation, another nested parallel loop is executed with different parameter values. Unlike WIEN2k, MeteoAG has an unbalanced structure, some parallel branches being longer than the others because of decision points in each parallel branch. To simulate the heterogeneous computing environment, we generated the processor speeds with a uniform distribution from 200 to 1000 MIPS with a step of 50. We created activity workloads based on existing historical data collected by running these applications using the ASKALON in the Austrian Grid during the last few years. We estimated the costs of resources based on Amazon prices, as summarized in Table 6. 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 [18] which approximates the clock frequency of a chip as the cube-root of its power consumption. Finally, we considered that resources are fully connected using a 10 gigabit Ethernet network.

TABLE 6. 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)

To investigate the impact of the application size on the efficiency of the algorithm, we categorized the workflows into three classes: *small*, *medium*, and *large* based on their number of activities (cf. Table 7). Workflows from each class are generated randomly using a uniform distribution within the specified ranges. We generated the user constraints randomly within their validity ranges too. WIEN2k and MeteoAG are computational-intensive applications with data transfers below 10 MB, thus we limited the data transfer between two activities to this threshold by assigning uniformly generated random values in the interval [0,10].

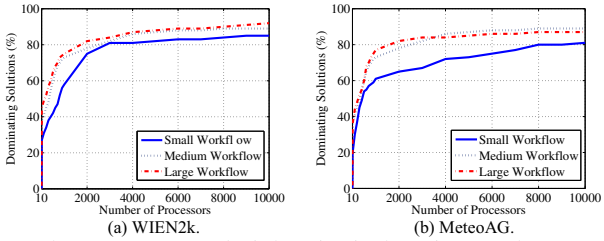
TABLE 7. WORKFLOWS CLASSES.

*Workflow**	*Small*	*Medium*	*Large*
WIEN2k	7—100	200—700	1000—2003
MeteoAG	71—1000	5000—10000	15000—20702

* *The number of workflow activities is based on the applications type.*

B. Experimental Results

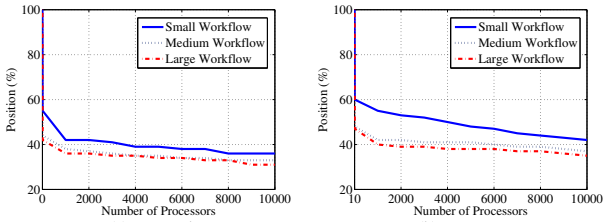
In the first part of the experiments, we investigated the percentage of solutions that dominate the constraint vectors. We run these experiments for 1000 workflows of various sizes from each class using a different number of processors. Fig. 4 shows that for a small number of processors, the MOLS algorithm often fails to find dominant solutions because there is not enough variety of resources to select. As soon as this number grows, the performance of the algorithm in finding dominant solutions substantially increases. Fig. 4 shows better results for WIEN2k than MeteoAG which is due to the balanced structure of WIEN2k compared to MeteoAG. Nevertheless, the impact of the workflow shape on the solution decreases with larger workflows and machine sizes.



(a) WIEN2k. (b) MeteoAG.

Figure 4. Percentage of solutions that dominate the constraint vector.

In the second part of the experiments, we evaluated the performance of MOLS in the absence of a constraint vector. In this case, the approach of the algorithm is to find a solution as close as possible to the Utopia point. To evaluate the quality of results, we defined 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: $Position = \frac{Distance(Solution,\ Utopia\ point)}{Distance(Nadir\ point,\ Utopia\ point)}$. This metric indicates how close the generated solutions are to the Utopia points (the lower, the better). Fig. 5 shows 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 Fig. 4, the results obtained 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.



(a) WIEN2k. (b) MeteoAG.

Figure 5. Quality of solutions.

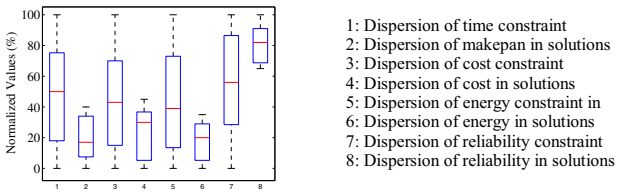
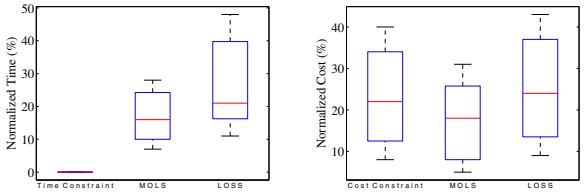


Figure 6. Dispersion of the constraints and solutions.

To assess the overall 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. 6, 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).

In the third part of the experiments, we compared MOLS with the LOSS algorithm [1] 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 tasks 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 task 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 tasks by considering the smallest values of the weight function for all tasks 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 time constraint equal to the minimum makespan for MOLS. All values have been normalized again.



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

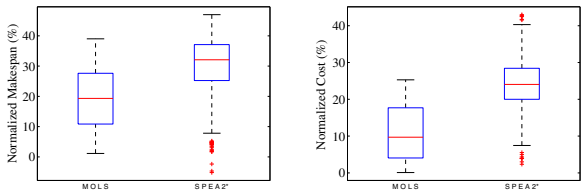
Figure 7. Dispersion of objectives for MOLS and LOSS.

As shown in Fig. 7a, the solutions generated by MOLS have in general shorter makespans compared to LOSS. The normalized time constraints in Fig. 7a are zero, since the time constraint in these experiments is the lower bound of the validity range. Fig. 7b 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 8 indicating that MOLS outperforms LOSS with respect to both objectives. The reason is that MOLS does not try to find the 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.

TABLE 8. THE AVERAGE MEASURED OBJECTIVES.

		MOLS			*LOSS*		
		Small	*Medium*	*Large*	*Small*	*Medium*	*Large*
WIEN2k	*Time (s)*	$1.8 \cdot 10^5$	$7.9 \cdot 10^5$	$1.4 \cdot 10^6$	$2.2 \cdot 10^5$	$8.3 \cdot 10^5$	$2.3 \cdot 10^6$
	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

In the last part of the experiments, we compared MOLS with a multi objective evolutionary algorithm called SPEA2* [19], which is based on the well-known SPEA2 algorithm [20]. SPEA2 is an a-posteriori multi-objective optimization technique [4] that approximates the entire Pareto set. SPEA2* defines two new fitness functions: one for the time constraint and another for the budget constraint. Each solution in SPEA2* is modeled by two strings: task assignment and execution string. Afterwards, the algorithm follows the SPEA2 algorithm and uses genetic operators to approximate the Pareto set. We used the SPEA2 source code available in the jMetal library [21] to implement the SPEA2* algorithm and experimented with the default setting proposed in [19] (e.g. length of the Pareto set to 10). We ran 100 experiments using the WIEN2k and MeteoAG workflows with different workflow and machine sizes. Similar to the prior experiments, we present normalized results. Fig. 8 displays the dispersion of the makespan and cost of MOLS and SPEA2* which indicate that the solutions of MOLS appear to be better. For a better perception of the solutions found by MOLS compared to the estimated Pareto frontiers of SPEA2 and the constraint vectors, Fig. 9 represents the result of two sample experimental run using WIEN2k. Similar to [19], we generated in each execution ten Pareto solutions using the SPEA2*. We can observe that the solutions of MOLS are dominating the constraint vector and also some of the Pareto points approximated by SPEA2*.



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

Figure 8. Dispersion of objectives for MOLS and SPEA2*.

Nevertheless, in some experiments the SPEA2* solutions dominate the MOLS ones. To investigate how often this situation happens, we use the *coverage* metric [22] 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. 10, the average coverage of MOLS is considerably larger than the average coverage of SPEA2*. Consequently, although SPEA2 is an efficient algorithm for many problems [20], these experiments prove that

MOLS is more suitable than the extended version of SPEA2 (SPEA2*) for workflow scheduling. The main reason for this superiority is that MOLS considers the activity precedence constraints in all phases of the algorithm.

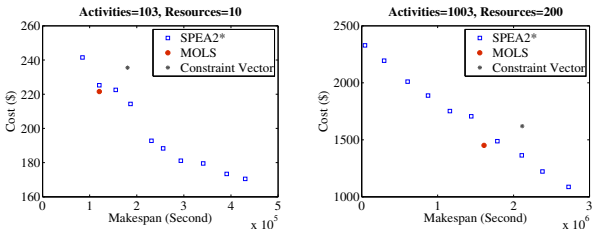
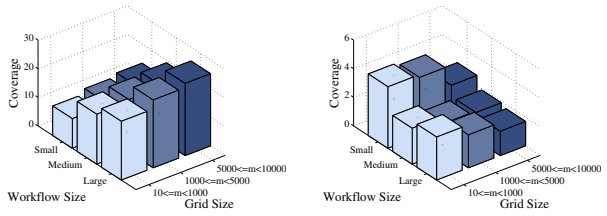


Figure 9. Objective spaces for two WIEN2k sample experiments.



(a) Coverage of MOLS on SPEA2*. (b) Coverage of SPEA2* on MOLS.

Figure 10. Average coverage of MOLS and SPEA2*.

VII. Related Work

Most related work is limited to two objectives and only a few efforts propose a solution for workflow-dependent tasks.

Two scheduling algorithms called LOSS and GAIN [1] schedule a DAG under a budget constraint based on a two-phase optimization process: the first phase optimizes one criterion, while in the second phase the budget constraint is applied. In [13], 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 overall 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 objective with respect to the defined constraint. Furthermore, they use a rescheduling phase that introduces important overhead that makes the scheduling process non-scalable.

A bi-criteria genetic optimization algorithm has been proposed in [2]. The fitness function is defined as a combination of the partial fitness functions of the objectives. The algorithm is budget constrained that considers no execution deadline.

The two workflow scheduling algorithms proposed in [3] (list and genetic-based heuristics) trade off execution time for reliability. The algorithms consider no constraints for the objectives and only concentrate on fairly optimizing the objectives without weighting them.

The list workflow scheduling heuristic in [23] targets makespan as the first objective and reliability as the second. The key idea of the approach is to replicate the activities on proper resources to gain both reliability and makespan optimization. The method is lexicographic, meaning that the objectives are arranged in the order of their importance. Constraints for the objectives are not considered.

The algorithm in [12] targets DAG execution time minimization while keeping the number of failures equal to a constant x. To this end, the algorithm replicates x+1 copies of each activity on different processors. The authors also suggest an extended algorithm which tries to improve the reliability of the whole systems which obviously suffers from redundancy.

The generic multi-objective approaches in [24] target task scheduling of different users. The algorithms are proposed for two situations: constant penalty function and time-invariant penalty function. Each user tries to increase an own utility, while the scheduler is responsible to increase the overall utility of the users by assigning them priorities. The algorithm is restricted to independent tasks.

The algorithm in [19] considers multi-objectives evolutionary algorithms to solve a general multi-objective workflow scheduling problem. The output is an approximation of the Pareto set, thus, selecting the proper solution from this set remains a problem. We demonstrated in our experiments that our proposed solution outperforms this algorithm.

In [25], the authors study the scheduling of pipeline workflows on homogenous platforms with respect to latency and throughput. The algorithm does not consider general workflows and assumes one constrained objective in each execution.

The approach proposed in our paper is distinct from the related work by simultaneously considering four constrained objectives. Our algorithm uses Pareto relations to find solutions that dominate or efficiently approach the user constraints. Based on [4], MOLS can be categorized as an a-priori multi-objective scheduling method looking for one single solution that satisfies the user's preferences and constraints. Determining the entire Pareto set is obviously not necessary.

VIII. Conclusion

In this paper we proposed an efficient multi-objective list scheduling algorithm for workflow applications in heterogeneous systems such as Grids and Clouds, implemented it in the ASKALON programming and execution environment. The algorithm follows a double strategy based on Pareto dominance relationships: maximize the distance to a user-defined constraint vector for dominant solutions and minimizing it otherwise. Extensive simulation experiments for two real-world applications with balanced and unbalanced structures demonstrate that, in most of cases, the obtained solutions dominate the user-specified constraints. In addition, the proposed algorithm outperforms a related bi-criteria scheduling heuristic and a bi-criteria genetic algorithm. An interesting future direction is to extend the MOLS algorithm for dynamic workflow scheduling that takes into account prediction and other model inaccuracies characteristic to distributed applications and environments.

Acknowledgment

The research was funded by the Austrian Science Fund (FWF): TRP 72-N23 and the Standortagentur Tirol: RainCloud.

REFRENCES

[1] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos, "Scheduling workflows with budget constraints," *in Integrated Research in GRID Computing, ser. CoreGRID*, Springer Verlag, pp. 189–202, 2007.

[2] J. Yu and R. Buyya, "A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms," *Workshop on Workflows in Support of Large-Scale Science*, HPDC 2006, IEEE CS Press, June 19-23, 2006.

[3] A. Dogan and F. Ozguner, "Trading off Execution Time for Reliability in Scheduling Precedence-Constrained Tasks in Heterogeneous Computing," *15th International Parallel and Distributed Processing Symposium*, p. 10060, 2001.

[4] R.T. Marler and J.S. Arora, "Survey of multi-objective optimization methods in engineering," *Structural and Multidisciplinary Optimization*, Vol. 26, No. 6, pp. 369-395, April 2004.

[5] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. L. Truong, A. Villaz´on, and M. Wieczorek, "Askalon: a grid application development and computing environment," *in 6th IEEE/ACM International Conference on Grid Computing*, pp. 122–131, November 13-14, 2005.

[6] K. Kurowski, J. Nabrzyski, A. Oleksiak and J. Węglarz, "Multicriteria Aspects of Grid Resource Management," *in Scheduling Jobs on the Grid – Multicriteria Approach, Grid Resource Management*, Kluwer, 2003.

[7] S. Ostermann, K. Plankensteiner, R. Prodan and T. Fahringer, "GroudSim: An Event-based Simulation Framework for Computational Grids and Clouds," *in CoreGRID/ERCIM Workshop on Grids and Clouds*, Springer, August 2010.

[8] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Gener. Comput. Syst.*, vol. 25, 3, pp.237-256, March 2009.

[9] H. Topcuouglu, S. Hariri, and M. you Wu, "Performance-effective and low- complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13 (3), 2002, 260–274.

[10] J. Pineau, Y. Robert, and F. Vivien, "Energy-Aware Scheduling of Flow Applications on Master-Worker Platforms," *in Proc. Euro-Par*, pp.281-292, 2009.

[11] S. U. Khan and C. Ardil, "Energy Efficient Resource Allocation in Distributed Computing Systems," *in International Conference on Distributed, High-Performance and Grid Computing (DHPGC)*, pp. 667-673, Singapore, 2009.

[12] A. Benoit, M. Hakem, Y. Robert. "Multi-criteria scheduling of precedence task graphs on heterogeneous platforms," *The Computer Journal*, 2009.

[13] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility Grids," *1st International Conference on e-Science and Grid Computing*, IEEE Computer Society Press, pp. 140–147, 2005.

[14] H. Mohammadi Fard, R. Prodan, G. Moser, T. Fahringer, "A Bi-Criteria Truthful Mechanism for Scheduling of Workflows in Clouds," *Third IEEE International Conference on Cloud Computing Technology and Science*, Cloudcom, pp. 599-605, 2011.

[15] F. Nadeem and T. Fahringer, "Predicting the execution time of grid workflow applications through local learning," *In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1-12, Portland, Oregon, ACM, 2009.

[16] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz, "WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties," *Institute of Physical and Theoretical Chemistry*, TU Vienna, 2001.

[17] 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,and J. P. McFadden, "RAMS 2001: Current status and future directions," *Meteorology and Atmospheric Physics*, vol. 82, pp. 5–29, 2003.

[18] 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," *EEE Micro,* vol 20(6), pp. 26–44, 2000.

[19] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on Grids," *in 8th International Conference on Grid Computing*, IEEE Computer Society Press, pp. 10–17, 2007.

[20] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization," *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems*, pp. 95–100. International Center for Numerical Methods in Engineering, 2002.

[21] J.J. Durillo, A.J. Nebro and E. Alba, "The jMetal Framework for Multi-Objective Optimization: Design and Architecture," *CEC 2010*, pp: 4138-4325. July 2010.

[22] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," *IEEE Transactions on Evolutionary Computation*, 3(4), pp. 257–271, 1999.

[23] I. Assayad, A. Girault, and H. Kalla, "A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints," *in International Conference on Dependable Systems and Networks*, p. 347, IEEE Computer Society Press, 2004.

[24] A. Dogan and F. Özgüner, "Scheduling of a meta-task with QoS requirements in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, 66 (2) , pp. 181–196, 2006.

[25] A. Benoit, V. Rehn-Sonigo and Y. Robert, "Multi-criteria scheduling of pipeline workflows," *International Conference on Cluster Computing*, pp.515-524, IEEE Computer Society , September, 2007.