2012 IEEE 4th International Conference on Cloud Computing Technology and Science

MOHEFT: A Multi-Objective List-based Method for Workflow Scheduling

Juan J. Durillo, Hamid Mohammadi Fard, Radu Prodan
Institute of Computer Science, University of Innsbruck
Innsbruck, Austria
{juan,hamid,radu}@dps.uibk.ac.at

***Abstract*—Nowadays, scientists and companies are confronted with multiple competing goals such as makespan in high-performance computing and economic cost in Clouds that have to be simultaneously optimized. Multi-objective scheduling of scientific workflows in distributed systems is therefore receiving increasing research attention. Most existing approaches typically aggregate all objectives in a single function, defined a-priori without any knowledge about the problem being solved, which negatively impacts the quality of the solutions. In contrast, Pareto-based approaches having as outcome a set of several (nearly-) optimal solutions that represent a tradeoff among the different objectives, have been scarcely studied. In this paper, we propose a new Pareto-based list scheduling heuristic that provides the user with a set of tradeoff optimal solutions from where the one that better suits the user requirements can be manually selected. We demonstrate the potential of MOHEFT for a bi-objective scheduling problem that optimizes makespan and economic cost in a Cloud-based computing scenario. We compare MOHEFT with two state-of-the-art approaches using different synthetic and real-world workflows: the classical HEFT algorithm used in single-objective scheduling and the SPEA2* genetic algorithm used for multi-objective optimisation problems.**

I. Introduction

Nowadays, many scientists and researches are moving towards Cloud computing for achieving high performance. This paradigm brings a new operational model where resources are managed by specialized data centers and rented only under demand and for the period of time they need to be used, is becoming very attractive for companies and institutions.

Scientific workflows are a popular way of modeling applications to be executed in parallel or distributed systems like Clouds. Once the workflow is composed, one of the most challenging research topics is how to schedule the different tasks onto the available resources. Traditionally, in parallel and distributed systems, workflow scheduling has been targeted to optimize the performance, measured in terms of the makespan or time of completing all tasks [12], [10]. This problem has been shown to be NP-complete. In the context of Cloud computing the user has to additionally care about the economical cost incurred by renting resources. Most of the commercial Clouds offer different types of resources at different prices. For example, in Amazon EC2 (http://aws.amazon.com/ec2/pricing/) a user can choose among four different types of resources, where the fastest one is eight times more expensive than the slowest. Therefore, workflow scheduling becomes a multi-objective optimization problem where no single optimal solution exists. For example, the tasks can be all scheduled on the fastest resources, thus involving a high economical cost, or on the cheapest resources with a reduced performance. Additionally, there may be several tradeoff solutions in between these two extreme cases.

In most related work [8], [9], [2], workflow scheduling optimizing several competing objectives has been simplified to a single-objective problem by aggregating all objectives in a single analytical function. The main drawback of these approaches is that the aggregation of the different objectives is made a-priori, with no knowledge about the workflow, infrastructure, and in general about the problem being solved. Therefore, the computed solution may not properly capture the user preferences. On the other hand, few approaches computing tradeoff solutions have been proposed, which have a main advantage over aggregative ones: the user is provided with a set of optimal solutions from which the one that better suits the requirement or preferences is manually selected.

In this paper, we introduce a new multi-objective workflow scheduling method called *Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)* as an extension to the well-known HEFT mono-objective workflow scheduling algorithm. This new proposal is a heuristic-based method that computes a set of tradeoff solutions (instead of a single one) with a small additional overhead compared to traditional single-objective methods. In doing this, MOHEFT builds several workflow schedules in parallel. Although there is no limitation in the number of objectives that MOHEFT can optimize simultaneously, we demonstrate its potential for optimizing makespan and economical costs in a Cloud computing system. We highlight the advantage of our method compared to two state-of-the-art approaches using workflows with different characteristics:

- HEFT [12], assuring that our method computes makespan solutions of the same quality or superior;
- SPEA2* [14], a genetic algorithm for computing tradeoff solutions in the multi-objective optimisation theory.

The rest of the paper is organized as follows. In the next section we describe the related work. Section III defines the abstract workflow, resource, and problem definition underneath our approach. Section IV gives a short background on multi-criteria optimisation required for a better understanding of our work. Section V presents our new multi-objective workflow scheduling algorithm, which is then evaluated for several synthetic and real-world workflows in Section VI. Finally, we

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

summarise the conclusions and the future work in Section VII.

II. Related Work

The most common multi-objective workflow scheduling technique combines the multiple objectives in a single function, for example by assigning different weights to the different objectives and optimizing the resulting aggregation function. The main differences in these approaches is the way in which preferences are expressed. For example, in [8] reliability (in terms of resource failures) and makespan are combined using a weight vector which is provided by the user. The same objectives are optimized in [9] and [2]. In the former, both objectives are given the same importance in the formulation (so the same preference for the goals is assumed). In the latter, the preferences are introduced by means of constraints over the different objectives. A general disadvantage these approaches is that the computed solution depends on the combination of the multiple objectives, which is done a-priori and without any information about the problem being solved. This fact implies that, if the aggregation function does not capture the user preferences in an accurate way, the computed solution may not be satisfactory for the solved problem. Additionally, the objective functions require to be normalized to the same interval to properly capture these preferences, which requires the optimal solutions in terms of each objective to be known.

Another way of combining the different objectives in a single function is by imposing user constraints to each objective (i.e. a desired threshold). Once the constraints have been set, the idea is to compute a solution optimized for one objective. After computing this solution, several modifications are made with the aim of improving according to another objective, as long as the constrains are not violated. An example work using this approach has been proposed in [11] for optimizing makespan and economical cost in utility Grids. The main weakness of this approach is that the number of objectives that can be optimized is limited to a few. Furthermore, it is required to establish an order in which the objectives are optimized, hence including some sort of preferential information. Finally, reasonable a-priori values for the constraints are often unknown until the first schedule is computed.

To the best of our knowledge, only a few techniques that compute a set of tradeoff solutions have been proposed. In [14], several genetic multi-objective algorithms are analyzed. The obtained results showed that these algorithms are able to compute good solutions but require a lot of computational time. To overcome this difficulty, the authors showed that including an already known good solution in terms of each of the objective significantly improves the convergence speed of the algorithm and the obtained results.

In [5], a multi-objective list-based heuristic is proposed and applied to optimizing makespan and reliability. In every iteration of the algorithm, a new set X of solutions is computed. As this set may get quite large, the number of solutions in every iteration is pruned in order to avoid an exhaustive search using a hypervolume metric which indicates the quality of a Pareto set (see Section IV). A main drawback of this approach is that computing the hypervolume is computationally expensive and exponential with the number of objectives. Additionally, the hypervolume has to be computed several times in every iteration degrading the performance.

The method proposed in this paper is also a multi-objective list-based heuristic. Our method extends HEFT for computing a set of tradeoff solutions, so we assure that at least one tradeoff solution as good as the one computed by HEFT in terms of makespan. In contrast to [5], we employ a new metric called crowding distance [6] of polynomial complexity with the number of solutions and the number of objectives to avoid the exhaustive search and and the computationally expensive hypervolume method for pruning the set of tradeoff solutions. To the best of our knowledge, this is the first multi-objective optimisation proposal that extends a list-based heuristic for workflow scheduling in a Cloud computing environment.

III. Model

This section formally describes the workflow, resource, and problem definition underneath our aproach.

A. Workflow Model

A *workflow application* is usually modeled as a directed acyclic graph (DAG), $W = (A, D)$ consisting of n tasks $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through 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 from activity A_i to activity A_j. We use $pred(A_i) = \{A_k |\, (A_k, A_i, Data_{ki}) \in D\}$ to denote the *predecessor* set of activity A_i, (i.e. activities to be completed before starting A_i). Finally, we assume that the computational workload of every activity A_i is known and is given by the number of machine instructions required to be executed.

B. Resource Model

We assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$. For every resource R_j we assume that we know its speed s measured in the number of instructions per second. Additionally, every resource has a price model consisting of four components:

1) pe_{R_i} is the price per second for using the resource;
2) ps_{R_i} is the price in MB per second for storing the data;
3) pi_{R_i} is the price for receiving the incoming data in terms of transferred MB;
4) po_{R_i} is the price for sending data to other resources in terms of transferred MB.

Without loss of generality, this price model can be adapted to match any of the existing commercial Clouds.

C. Problem Definition

Our problem consists in scheduling the execution of the workflow tasks on the resources in such a way that the makespan and the economical costs are minimized. In the rest of this paper, we will use $sched(A_i)$ to denote the resource on which task A_i is scheduled to be executed.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

1) Makespan: For computing the workflow makespan, it is first necessary to define the *execution time* $t_{(A_i,R_j)}$ of an activity A_i on a resource $R_j = sched\,(A_i)$ as the sum of the time required for transferring the biggest input data from any $A_p \in pred(A_p)$ and the time required to execute A_i in R_j:

$$t_{(A_i,R_j)} = \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{jq}} \right\} + \frac{workload\,(A_i)}{s_j}, \quad (1)$$

where $Data_{pi}$ is the size of the data to be transferred between A_p and A_i, b_{pj} is the bandwidth of one TCP stream between the resource where task A_p was executed and the resource R_j, $workload(A_i)$ the length of the task A_i in machine instructions, and s_j the speed of the resource R_j in number of machine instructions per second. Next, we can compute the *completion time* T_{A_i} of activity A_i considering the execution time of itself and its predecessors, as follows:

$$T_{A_i} = \begin{cases} t_{(A_i,sched(A_i))}, & pred\,(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \left\{ T_{A_p} + t_{(A_i,sched(A_i))} \right\}, & pred\,(A_i) \neq \emptyset. \end{cases} \quad (2)$$

The workflow makespan is finally defined as the maximum completion time of all the activities in the workflow:

$$T_W = \max_{i \in [1,n]} \left\{ T_{(A_i,sched(A_i))} \right\}. \quad (3)$$

2) Economic Cost: We define the economical cost $C_{(A_i,R_j)}$ of executing an activity A_i on a resource R_j as the sum of the four cost components introduced in Section III-B:

$$\begin{aligned} C_{(A_i,R_j)} = \; & t_{(A_i,R_j)} \cdot pe_{R_j} + Data\,(A_i) \cdot t_{(A_i,R_j)} \cdot ps_{R_i} + \\ & + In(A_i) \cdot pi_{R_i} + Out\,(A_i) \cdot po_{R_i}, \end{aligned} \quad (4)$$

where $Data\,(A_i)$ represents the total data storage required for executing the activity A_i (including input, output, and temporary data) and $In\,(A_i)$ / $Out\,(A_i)$ is the sum of the data sizes transferred to / from A_i from / to activities executed on resource other than R_j. The economical cost of executing the entire workflow W the simple sum of executing all its tasks:

$$C_W = \sum_{i=1}^{n} C_{(A_i,R_j)} \quad (5)$$

IV. Multi-Objective Optimization Background

In this section, we introduce concepts from the *multi-objective optimization* theory for a better understanding of this work. We assume without loss of generality that minimization is the goal for all the objectives, as any maximization problem can be defined in terms of a minimization too.

A general, multi-objective optimisation problem can be formally defined as follows: find all the vectors $\vec{x} = [x_1, x_2, \ldots, x_n]$ which minimize the vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_t(\vec{x})]^T$. For our particular problem, n represents the cardinality of the task set A ($n = |A|$) and the i-th component of a solution $\vec{x}$ represents the resource where task A_i is going to be executed: $sched\,(A_i) = x_i, x_i \in R$. In our bi-objective optimisation case, we have $t = 2$, where $f_1(\vec{x})$ represents the makespan and $f_2(\vec{x})$ the economical cost.

As it is not possible to find a solution minimizing makespan and economical cost simultaneously, we need to introduce the concept of *dominance*. A solution $\vec{x_1}$ dominates a solution $\vec{x_2}$, if the makespan and economical cost of $\vec{x_1}$ are smaller than those of $\vec{x_2}$. Conversely, two solutions are said to be non-dominated whenever none of them dominates the other (i.e. one is better in makespan and the other is better in economical cost). In Figure 1 for example, the solution labelled a dominates the one labelled b because it has better makespan and economical cost. Similarly, a dominates c too. Meanwhile, a and d are non-dominated because a is better in makespan, but d is better in economical cost. A set of non-dominated solutions is called *Pareto set* (the trend line containing the a, d, and e solutions) and represents a set of tradeoff solutions among the different objectives. Every solution in this set represents a different mapping of the workflow tasks with different makespans and economical costs.

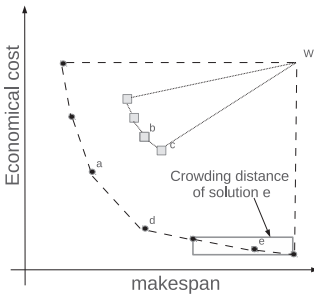

Fig. 1. Comparison of multi-objective tradeoff solutions.

A Pareto front can be seen as a tool for decision support and preferences discovery. Its shape can provide insight to researches or scientists (from now decision makers), allowing them in many cases to explore the possible space of non-dominated solutions with certain properties, and possibly revealing regions of particular interest which cannot be seen until the Pareto front is known. In this way, the users do not have to set their preferences before finding a solution, instead the preferences are discovered afterwards. To this end, not all the Pareto fronts are valid. A good Pareto front is one which provides accuracy (solutions close the optimal ones) and diversity (uniformly cover all the possible ranges of optimal solutions). A way of measuring the quality of a set of tradeoff solutions is the *hypervolume*. Given a set of tradeoff solutions X, the hypervolume $HV(X)$ measures the area enclosed between the points in X and a reference point W (see Figure 1), usually selected as the maximum objective value (e.g. between the makespan and economical cost). This way, the better the points contained in X and the most diverse they are, the higher $HV(X)$. In Figure 1 for example, the set containing the solid round points is better than the set containing the squared solutions because the area enclosed within the dashed lines is larger than the one represented by the dotted lines.

V. MOHEFT: Multi-Objective Heterogenous Earliest Finish Time Algorithm

In this section we describe our proposed multi-objective scheduling algorithm for computing a set of tradeoff solutions (instead of a single one) as an extension to the HEFT list scheduling algorithm. For a better understanding, we start by describing the mono-objective version of the algorithm and extend it afterwards for dealing with multiples objectives.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

Algorithm 1 HEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Ensure: $sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Workflow schedule
1: **function** HEFT(W,R)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: $sched_W \leftarrow \emptyset$ ▷ Initialize workflow schedule with empty set
4: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
5: $T_{\min} \leftarrow \infty$
6: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
7: $T_{Rank_i} \leftarrow \max_{A_p \in pred(Rank_i)} \left\{T_{A_p} + t_{(Rank_i, R_j)}\right\}$ ▷ Compute completion time of $Rank_i$
8: **if** $T_{Rank_i} < T_{\min}$ **then** ▷ Save the minimum completion time
9: $T_{\min} \leftarrow T_{Rank_i}$
10: $R_{\min} \leftarrow R_j$
11: **end if**
12: **end for**
13: $sched_W \leftarrow sched_W \cup (Rank_i, R_{\min})$ ▷ Schedule the task
14: **end for**
15: **return** $sched_W$
16: **end function**

Algorithm 2 MOHEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Require: K ▷ Number of tradeoff solutions
Ensure: $S = \bigcup_{i=1}^{K} sched_W, sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Set of K tradeoff workflow schedules
1: **function** MOHEFT(W,R,K)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: **for** $k \leftarrow 1, K$ **do** ▷ Create K empty workflow schedules
4: $S_k \leftarrow \emptyset$
5: **end for**
6: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
7: $S' \leftarrow \emptyset$
8: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
9: **for** $k \leftarrow 1, K$ **do** ▷ Iterate over all tradeoff schedules
10: $S' \leftarrow S' \cup \{S_k \cup (Rank_i, R_j)\}$ ▷ Add new mapping to all intermediate schedules
11: **end for**
12: **end for**
13: $S' \leftarrow$ SORTCROWDDIST(S', K) ▷ Sort according to crowding distance
14: $S \leftarrow$ FIRST(S', K) ▷ Choose K schedules with highest crowding distance
15: **end for**
16: **return** S
17: **end function**

A. HEFT: Heterogeneous Earliest Finish Time Algorithm

The Heterogeneous Earliest Finish Time Algorithm (HEFT) is a popular list-based heuristic scheduling algorithm for optimizing the makespan [12] in workflow applications, described in pseudocode in Algorithm 1. The method consists of two phases: ranking and mapping. In the ranking phase (line 2), the order in which the activities are being mapped is computed using the B-rank metric (distance of the activity to the end of the workflow). The idea of this ranking is to execute before those tasks having more dependent tasks than others. Further details about how to sort the tasks can be found in [12]. Once the execution order is determined, the second phase consists in assigning each task to the resources (lines 4–14) following the order computed in the first phase. For each task (line 4) and for each resource (line 6), the completion time of that task on that resource is computed (line 7). Finally, the task is mapped onto the resource where it is finished earlier (line 13). After all tasks have been mapped, the workflow schedule is returned (line 15).

B. MOHEFT: Multi-Objective Heterogenous Earliest Finish Time Algorithm

As described before, HEFT builds a solution by iteratively mapping tasks onto resources. That mapping is aimed at minimizing the completion time of every single task, so in every iteration only the resource which minimizes this goal is considered. When multiples objectives are considered, the goal is to compute a set of tradeoff solutions. To this end, we must allow the creation of several solutions at the same time instead of building a single solution. Additionally, instead of mapping every task onto the resource where it is finished earlier, we should allow mapping of tasks also to resources that provide a tradeoff between the considered objectives.

The MOHEFT algorithm extends HEFT with these ideas, as depicted in pseudocode in Algorithm 2. The only additional input parameter of MOHEFT is the desired size of the set of tradeoff solutions K. Our method also ranks the tasks using the B-rank (line 2). After that, instead of creating an empty solution as in HEFT, it creates a set S of K empty solutions (lines 3–5). Afterwards, the mapping phase of MOHEFT begins (lines 6–15). MOHEFT iterates first over the list of tasks (line 6) sorted by execution order. The idea is to extend every solution in S by mapping the next task to be executed onto all possible resources creating m new solutions. We store all the created solutions in a temporal set S' which is initially empty (line 7). For creating these new solutions, we iterate over the set of resources (line 8) and the set S (line 9), and add the new extended intermediate schedules to the new set S' (line 10). This strategy results in an exhaustive search if we do not include any restrictions, therefore MOHEFT only saves the best K tradeoffs solutions from the temporary set S' to the set S (lines 13–14). We consider that a solution belongs to the best tradeoff if it is not dominated by any other solution and if it contributes to the diversity of the set. For this last criterion, we employ the *crowding distance* defined in [6] and graphically depicted in Figure 1, which gives a measure of the area surrounding a solution where no other tradeoff solution is placed. Our criterion is to prefer solutions with a higher crowding distance, since this means that the set represents a wider area of different tradeoff solutions. After assigning all the tasks (line 16), the algorithm returns the set of K best tradeoff solutions.

C. Complexity

Given a set of n activities and m resources, the computational complexity of HEFT is $O(n \cdot m)$. MOHEFT only introduces two main differences with respect to HEFT: the creation of several solutions in parallel, and the possibility of considering resources providing a tradeoff solution. Considering that the set of tradeoff solutions is K, the complexity of MOHEFT is $O(n \cdot m \cdot K)$. Usually, the number of tradeoff solutions is a constant much lower than n and m. For example, a workflow can be composed of thousands of tasks and the set of tradeoff solutions can be accurately represented with tens of solutions. Thus, the complexity can be considered to be almost $O(n \cdot m)$, as in HEFT.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

VI. Evaluation

In this section we describe the experiments carried out for evaluating MOHEFT. We evaluated the suitability of our method for scheduling workflows with different properties by conducting hundreds of different experiments. Therefore, we have discarded the use of any commercial Cloud due to the entailed economical costs and relied on simulations using the GridSim [4] toolkit.

As far as the resource infrastructure is concerned, we simulated two different scenarios:

- *scenario 1* where the number of resources ranges between 10 and 100;
- *scenario 2* where the number of resources are similar to the number of tasks in the workflow.

We generated the speed and price of each resource using a random uniform distribution limited within a maximum and minimum speed. In particular, the speed ranges then between 10000 and 160000 mips. The prices for every resource is proportional to that velocity plus a fix term, and ranges between 0.052 cent and 0.082 per second of usage.

Alongside HEFT, we compared MOHEFT with the SPEA2* algorithm from the multi-objective optimisation theory. SPEA2 is a genetic algorithm proposed by Zitzler et al. [15] which works by mimicking the evolution of species in the nature. The algorithms works with a population of candidate solutions which are iteratively recombined with the aim of producing better ones. SPEA2* is a version of SPEA2 proposed in [14]. The main difference is that the algorithm is initialized with a nearly-optimal solution in terms of makespan (for example computed using HEFT) and in terms of economical cost (computed with another heuristic). In our work, we consider SPEA2* with a population size equal to the number of tradeoff solutions K that we want to compute and run the algorithm for 1000 generations. We implemented this algorithm using the jMetal framework [7]. In all the cases, we considered that the final size of the set of tradeoff solutions is $K = 10$.

We analyse and compare the results of the MOHEFT, HEFT, and SPEA2* in terms of three different criteria: the shortest makespan, the lowest economical cost, and the quality of the computed tradeoff solutions using the hypervolume metric introduced in Section IV.

We have simulated two kinds of workflows:

- *synthetic* using different types of artificially-generated properties;
- *real-world*, inspired from our real-world collaborations with scientists in the Austrian Grid.

Our experimental setup is summarized in Table I.

A. Synthetic Workflows

For the sake of generality, we started our evaluation on synthetic workflows of three different types using a workflow generator as the one described in [13] (see Figure 2):

- *type 1* where the number of tasks that can be executed in parallel ranges between one and two;
- *type 2* where the number of tasks that can be executed in parallel is high, and the workflow is balanced (same number of tasks in every level);

TABLE I
Experimental setup

Workflow	*Number of*			
	instances	*tasks*	*resources*	
Wien2k	100	100 – 1000	*Scenario 1* 10 – 100	*Scenario 2* 100 – 1000
Povray	100	100 – 1000	*Scenario 1* 10 – 100	*Scenario 2* 100 – 1000
Type 1	100	100 – 100	*Scenario 2* 100 – 1000	
Type 2	100	100 – 100	*Scenario 2* 100 – 1000	
Type 3	100	100 – 100	*Scenario 2* 100 – 1000	

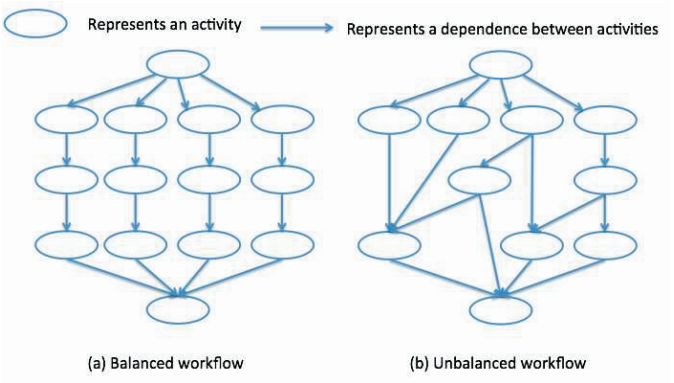

Fig. 2. Balanced and unbalanced workflows.

- *type 3* where the number of tasks that can be executed in parallel is high, but the workflow is unbalanced (the number of tasks in every level is different).

We generated the length of every activity and the data produced using a Gaussian distribution. For every type we considered 100 different instances having between 100 and 1500 different tasks and we evaluated them in *scenario 2*. Table I summarizes the experiments carried out.

1) Type 1: The obtained results for this type of workflow are depicted in Figure 3. If we compare the makespan and economical cost of the three analyzed methods (see Figure 3, left and middle columns) we observe that all methods have computed the same makespan solution, while for economical cost the results of HEFT were worse than those of MOHEFT and SPEA2*. This result demonstrate that MOHEFT does not degrade the performance in compared to the mono-objective HEFT. In the case of SPEA2*, the result is not a surprise since the algorithm is initialized with a good solution in terms of makespan and cost. Regarding to the quality of the tradeoff set, we observe that MOHEFT and SPEA2* computed solutions with similar HV value. This fact indicates that both methods perform the same when trying to find a schedule for this type of workflows.

2) Type 2: Also for the second type of workflows, the makespan and economical cost behave as in the previous cases, as it can be observed in Figure 4 (left and middle columns).

In terms of the quality of the set of tradeoff solutions, MOHEFT has outperformed SPEA2* in all the evaluated instances. In particular, the higher the number of activities in the workflow, the higher the difference of the quality of

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

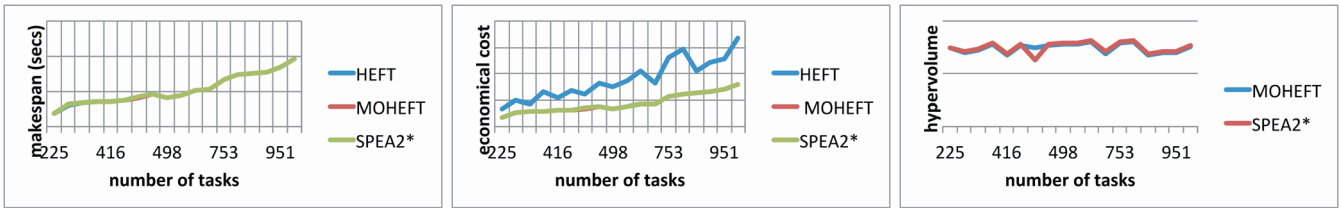

Fig. 3. Synthetic workflows of *type 1* with a low degree of parallelism.

solutions computed by these two techniques. This means that when a high number of activities can be executed in parallel, MOHEFT is able of producing better tradeoff solutions than SPEA2*. This result points out that in this kind of workflows the genetic operators designed for SPEA2* are not able of improving on the quality of the tradeoff solutions as much as MOHEFT is able to do.

3) Type 3: For the third type of workflows, MOHEFT is able of outperforming the results computed of HEFT and SPEA2* in terms of makespan, as observed in Figure 5. The explanation is that HEFT is a greedy heuristic and, hence, in every iteration it selects the resource that minimizes the makespan of the next task to execute. MOHEFT, on the contrary, builds several solutions in parallel achieving a better exploration of the search space. In terms of cost, the the results are similar to the previous analyzed workflows: MOHEFT and SPEA2* computed the best solutions, and the improvements compared to HEFT are higher when the number of activities in the workflow increase. Finally, the quality of the set of tradeoff solutions depicted in Figure 5 (right column) show that also for this type of workflows MOHEFT also outperformed SPEA2. As happened in workflows of Type 2, the differences between the two methods increase with the number of activities on the workflow.

B. Real-World Applications

In this section, we complete our validation on two real-world workflow applications: Wien2k and POV-Ray. These applications can be modeled with different parallelization sizes involving different number of activities. In this work we considered 100 different instances, each of them having between 100 and 1500 activities.

1) Wien2k: Wien2k [3] is a material science workflow for performing electronic structure calculations of solids using density functional theory based on the full-potential (linearized) augmented plane-wave ((L)APW) and local orbital (lo) method. WIEN2k is a workflow of *type 2* consisting of two parallel sections with sequential synchronisation activities in between (see Figure 7). The results obtained for the Wien2k workflow are depicted in Figure 6 for *scenario 1* (top) and *scenario 2* (bottom). In both cases, MOHEFT has been able to compute a solution with the same makespan as HEFT (Figure 6, left column) and also SPEA2*.

Finally, we observe in Figure 6 (right column) that the quality of the set of tradeoff solutions computed by MOHEFT has been always better than the ones computed by SPEA*.

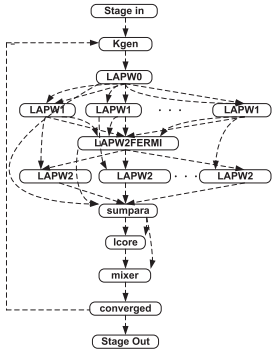

Fig. 7. Wien2K workflow.

2) POV-Ray: The Persistence Of Vision Raytracer (POV-Ray) [1] is a free tool for creating three-dimensional graphics, which is known to be a time consuming process used not only by hobbyists and artists, but also in biochemistry research, medicine, architecture and mathematical visualisation. We modeled a POV-Ray rendering scenario as a *type 2* workflow depicted in Figure 8 (left), where the description of a movie can be separated in several scenes, each scene being composed of several frames which can be rendered as parallel activities. Finally, all frames are merged into an `mpeg` movie. The results obtained for the POV-Ray workflow summarized in Figure 9 are similar to the ones obtained for Wien2k. First, the three methods have computed again the same solution for the best makespan. Second, MOHEFT and SPEA2 have provided the best results in terms of the economical cost, being HEFT again the worst alternative. As happened before, the higher the number of activities the higher the differences between the economical cost provided by MOHEFT and HEFT. Finally, the analysis of the hypervolume verifies that also in this case MOHEFT has also outperformed to SPEA2*. For *scenario 2* we can further observe that the higher the number of tasks to schedule, the higher the differences in the quality of results. An example of the set of tradeoff solutions computed by these methods for the same workflow instance is depicted in Figure 8 (right). As we can see, for every solution computed by SPEA2* it is possible to find a solution computed by MOHEFT with better makespan and better economical cost.

VII. Conclusions and Future Work

In this paper we proposed a new multi-objective list-based scheduling algorithm for scientific workflows called MOHEFT. In contrast to related work based on a-priori aggregative functions, MOHEFT is a truly multi-criteria optimisation algorithm aiming to approximate the Pareto set using a number of

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

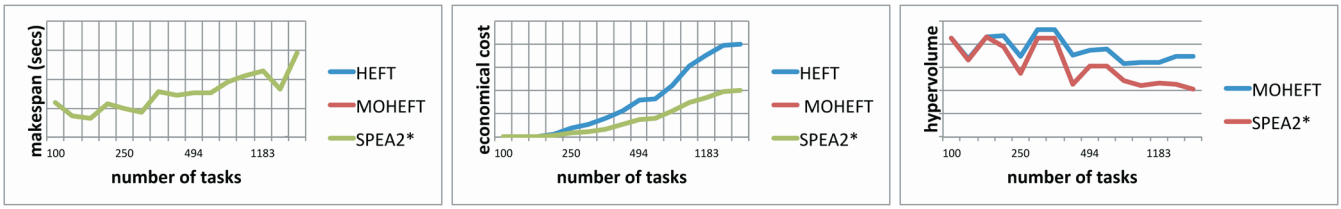

Fig. 4. Synthetic workflows of *type 2* with a high degree of parallelism and balanced structure.

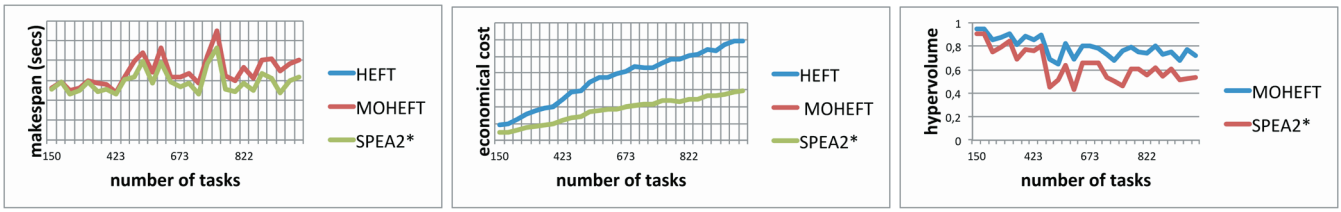

Fig. 5. Synthetic workflows of *type 3* with high degree of parallelism and unbalanced structure.

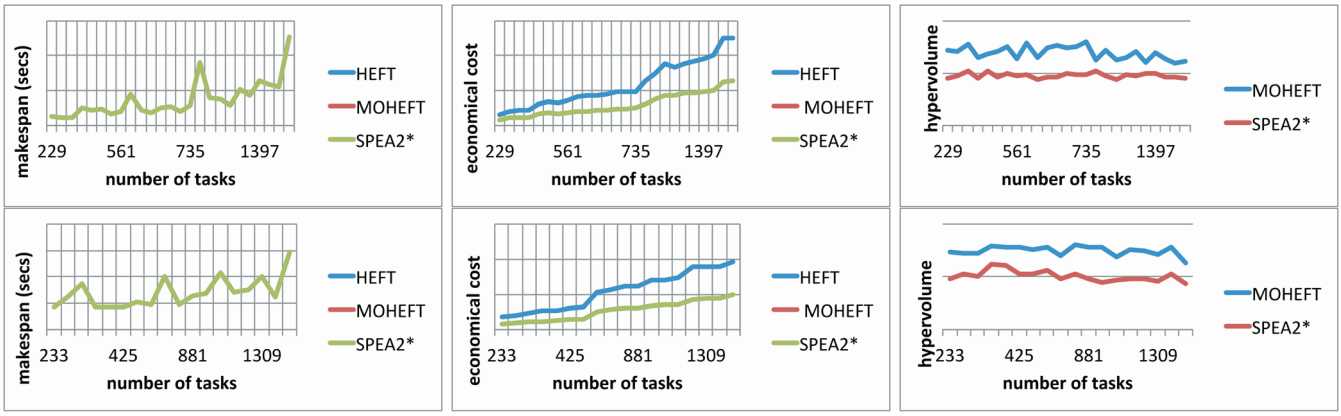

Fig. 6. Wien2K workflow results.

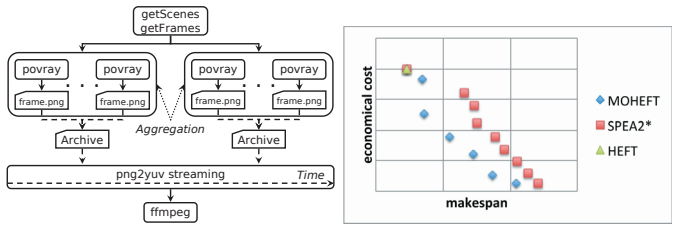

Fig. 8. POV-Ray workflow (left) and comparison of two tradeoff solutions (right).

high-quality tradeoff solutions (given as input) evaluated using a new crowding distance metric. We compared our method with one of the most popular workflow list scheduling algorithms (HEFT) and another state-of-the-art algorithm from the multi-criteria optimisation theory (SPEA2*). We considered in our evaluation both synthetic workflows using different characteristics, as well as two real-world ones. In all experiments, MOHEFT obtained equal or even better makespan as the mono-objective HEFT algorithm and better economical cost. For workflows with a few parallel activities, MOHEFT and SPEA2* computed solutions of the same quality having the same makespan as HEFT and better economical cost. For workflows with a high number of parallel activities and a balanced structure, MOHEFT outperformed to SPEA2*, the difference increasing with the number of workflow tasks. For workflows with a high number of independent tasks and unbalanced structure, MOHEFT delivered better makespan results than HEFT and SPEA2* and higher quality tradeoff solutions than SPEA2*. In future work we will implement the MOHEFT algorithm in the ASKALON environment and carry out experiments in commercial Clouds such as Amazon EC2. We will also include of more objectives in our optimization such as energy consumption and reliability.

References

[1] http://www.povray.org/.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, Firenze, Italy, June 2003. IEEE.

[3] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. Wien2k: An augmented plane wave plus local orbitals program for calculating crystal properties. Technical report, Institute of Physical and Theoretical Chemistry, TU Vienna, 2001.

[4] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, 14(13):1175–1220, 2002.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

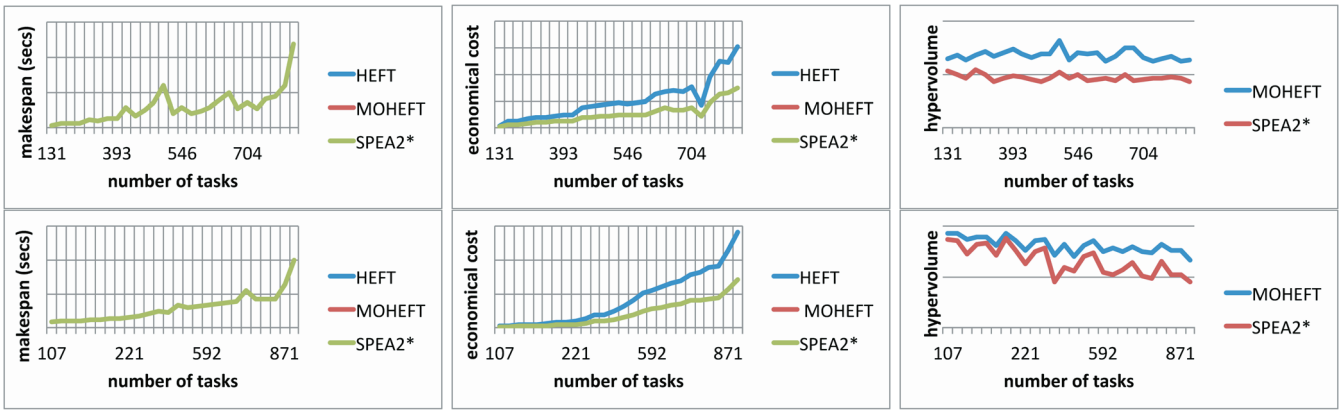

Fig. 9. POV-Ray workflow results.

[5] Louis-Claude Canon and Emmanuel. Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines. International Heterogeneity in Computing, 2011.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[7] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[8] Saurabh Kumar Garg, Rajkumar Buyya, and H. J. Siegel. Scheduling parallel applications on utility grids: time and cost trade-off management. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91*, ACSC '09, pages 151–160, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[9] Mourad Hakem and Franck Butelle. Reliability and scheduling on systems subject to failures. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, pages 38–, Washington, DC, USA, 2007. IEEE Computer Society.

[10] E. Ilavarsan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2):94–103, 2007.

[11] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D. Dikaiakos. Scheduling workflows with budget constraints. In *in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series*. Springer-Verlag, 2007.

[12] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260 –274, mar 2002.

[13] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In F. Xhafa and A. Abraham, editors, *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 109–153. Springer Berlin, 2008.

[14] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 10–17, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE