Workflow Scheduling in Amazon EC2

Juan J. Durillo[1], Radu Prodan[1], and Weicheng Huang[2]

[1] Institut fur Informatik,University of Innsbruck Austria
{juan,radu}@dps.uibk.ac.at

[2] National Center of High-performance Computing, National Applied Research Laboratories No. 7, Hsinchu, Taiwan
whuang@nchc.narl.org.tw

Abstract. Workflow scheduling has been traditionally targeted to map the execution of set of tasks onto a set of resources for makespan minimization. With the increasing popularity of Cloud computing systems, the financial cost entailed for executing these tasks plays also an important role. Existing works have however combined both, makespan and cost, on a single function and no analysis of the tradeoff between both criteria has been produced. In addition, no work in the context a real commercial cloud system exists. This paper includes a comparison of two real multi-objective workflow scheduling, MOHEFT and SPEA2*, in the context of Amazon EC2. The carried experiments show that MOHEFT outperforms SPEA2*, and that the analysis of the tradeoff solutions can help in selecting good scheduling solutions.

1 Introduction

Scientific workflows are an attractive model for building large scale applications. Typically, a workflow application consists of several (legacy) programs (referred from now on as tasks or activities) in the form of a dependency graph, where the input of some of these programs may depend on the output of the others. The workflow paradigm does not impose any restriction about the activities that compose an application, being possible that they belong to different actors or agents. In such a situation, a Distributed Virtual Environment (DVE) [10] appears as the ideal candidate for the execution of collaborative workflow applications.

Cloud computing is an approach to offer a DVE. Previous work [14] has pointed out, however, that one of the main challenges in the case of a cloud-based DVE solution has to do with the design of algorithms for resource provisioning, i.e., how many resources are required for executing an application. In the context of workflow-applications, the answer to the aforementioned issue depends on how the execution of the tasks composing a workflow is scheduled. Traditionally, that scheduling has been targeted to minimize the time required for executing the workflow, a.k.a makespan.

In a Cloud scenario the *economic cost* plays an important role on the scheduling. Many of commercial Clouds apply a hour-based pricing model, i.e., users are charged per hours of computation. This model introduces a tradeoff between makespan and economic cost. For example, a workflow that can sequentially run in less than an hour in a single machine may be accelerated by considering a parallel execution on several machines, thus increasing the price but decreasing the utilisation of the rented resources. In addition to this, some commercial clouds offer heterogeneous types of resources at different prices and with different performance. For example, in Amazon EC2 (`http://aws.amazon.com/ec2/pricing/`) a user can choose among different types of instances, where the fastest resource is about eight times more expensive than the slowest one. Under these circumstances, it is clear that workflow scheduling in a Cloud scenario is a multi-objective optimisation problem (MOP) aimed at optimising at least two conflicting criteria: makespan and economic cost. The main characteristic of MOPs is that no single solution exists that is optimal with respect to all objectives, but a set of tradeoff solutions known as *Pareto front*. Solutions within this set cannot be further improved in any of the considered objectives without causing the degradation of at least another objective.

While only a few approaches exist for computing these sets of tradeoff solutions, most of the related works [8,9,2] aggregate the different optimisation criteria in a single analytical function. The main drawback of this approach is that only a single solution, instead of a set, is computed. In addition, approaches computing the whole set of tradeoff solutions have been only applied in the context of utility grids or clusters.

The purpose of this paper is to compare and analyse the solutions computed by two multi-objective workflow scheduling methods on the context of a real commercial Cloud, Amazon EC2. On the one hand, we consider MOHEFT, a multi-objective list-based heuristic which extends the mono-objective workflow scheduing HEFT. Previous work [6] has demonstrated that MOHEFT can compute a high-quality solutions in a set of synthetically defined problems . On the other hand, we consider SPEA2* [17], as an application of evolutionary computation to solve this problem. Both, MOHEFT and SPEA2*, are applied in this work for scheduling two real workflow applications.

The paper is organised as follows. The next section describes the related work. Section 3 includes some definitions for a better understanding of this work. In Section 4, we describe MOHEFT and SPEA2*. We present in Section 5 the experimental setup for evaluating these techniques on Amazon EC2 (Section 6). Finally, we summarise the conclusions and the future work in Section 7.

2 Related Work

Many existing approaches to multi-objective workflow scheduling reduce the problem to mono-objective optimisation and compute only a single solution. There exist two different ways to do so. The first consists in aggregating all the optimisation criteria in a single function by means of using user preferences. Previous work using this approach combine reliability and makespan [2,8,9]. The second line consists in sorting and constraining the different criteria, which are later on optimised in a sequential fashion as much as possible without violating the imposed constraints. An example of work following this approach is [15] where the authors optimise for makespan and economic cost in utility Grids.

Approaches computing the whole Pareto front are also classified on two groups: evolutionary-based methods and list-based heuristic. Examples of evolutionary algorithms applied for multi-objective workflow scheduling are SPEA2*, NSGA-II*, and PAES* [17], a bi-objective genetic algorithm proposed in [12], and R-NSGA-II [7]. Different multi-objective extensions of list-based heuristic are proposed in [4,6] by extending HEFT [16], a popular heuristic aimed at optimising makespan of workflows in heterogeneous systems.

The aforementioned approaches were however always applied in the context of utility Grids. To the best of our knowledge, no application of them in a real commercial Cloud scenario exists. This work differs from related work in this sense. In particular, our target here is the application and comparison of MOHEFT and SPEA2* for workflow scheduling in Amazon EC2.

3 Model

3.1 Workflow Model

We model a *workflow application* as a directed acyclic graph: $W = (A, D)$ consisting of n activities $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through control flow and data flow dependencies; $D = \{(A_i, A_j, Data_{ij}) \mid (A_i, A_j) \in A \times A\}$, where $Data_{ij}$ represents the size of the data which needs to be transferred from activity A_i to activity A_j. We use $pred(A_i) = \{A_k \mid (A_k, A_i, Data_{ki}) \in D\}$ to denote the *predecessor* set of activity A_i, (i.e. activities to be completed before starting A_i). 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.

3.2 Resource Model

We assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$, which can be of any type as provided by Amazon EC2. In particular, we consider in this paper the five Amazon EC2 resources analysed in [1]. For a given resource R_j of a certain type, we know its average performance measured in GFLOPs and its price per every hour of computation (see [1]). The final price is based not only on the resources' usage, but also in the data stored and transferred among different instances which depends on four components: (1) price per hours of resource's usage PE_{R_i}; (2) price per MB of data storage PS_{R_i}; (3) price per MB of data received PI_{R_i}; (4) price per MB of data sent PO_{R_i}. The prices of these components depend on the Cloud provider.

Amazon EC2 introduces a constraint that must be included in the resource model. While in theory a user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of $N = 20$ instances that can be simultaneously acquired. These N resources can be of any type and do not have to keep invariant during the workflow execution.

3.3 Problem Definition

We use $sched(A_i)$ to denote the resource on which the task A_i is scheduled to be executed. We describe in the following how makespan and cost are computed.

Makespan. For computing the makespan, it is 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:

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

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:

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

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

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

Economic Cost. The economic cost depends on two terms: the computation cost $C^{(comp)}$ and the cost of data transfer and storage $C^{(data)}$.

We define $C^{(data)}_{(A_i,R_j)}$ as the cost of the data transfers $In(A_i)$ and $Out\,(A_i)$ and storage $Data\,(A_i)$ resulting from executing activity A_i on resource R_j:

$$C^{(data)}_{(A_i,R_j)} = Data\,(A_i) \cdot t_{(A_i,R_j)} \cdot PS_{R_i} + In(A_i) \cdot PI_{R_i} + Out\,(A_i) \cdot PO_{R_i}, \tag{4}$$

In defining the cost $C^{(comp)}_{R_j}$ of using a resource R_j, we assume that for each task A_i executed on R_j we record two timestamps: $t^{(start)}_{A_i}$ when the activity starts and $t^{(end)}_{A_i}$ when the activity finishes its execution. We consider without loss of generality that the times for transferring the input $In\,(A_i)$ and the output data $Out\,(A_i)$ are included in the interval between $t^{(start)}_{A_i}$ and $t^{(end)}_{A_i}$.

Let us consider now the set of all p activities scheduled on resource R_j denoted as $\{J_1,\ldots,J_p\}$, where $p < n$ and $sched\,(J_i) = R_j, i \in [1,p]$, sorted based on their start timestamp: $t^{(start)}_{J_1} < \ldots < t^{(start)}_{J_p}$. Based on this ordering, we cluster these activities in $q \leq p$ different *groups* $G^{(j)}_k$, $1 \leq k \leq q$, so that all activities in one group are executed consecutively without releasing the resource. After the activity with the largest start timestamp in the group completes, the resource is released.

We construct the first group $G_1^{(j)} = \{J_1, \ldots, J_r\}, r \leq p$ following three rules:

1. The first activity J_1 belongs to the first group: $J_1 \in G_1^{(j)}$;
2. Every activity $J_i \in G_1^{(j)}, 2 \leq i \leq r$, starts before the current leased hour expires and before the machine is released:

$$t_{J_i}^{(start)} < t_{J_1}^{(start)} + \left\lceil \frac{t_{J_{i-1}}^{(end)} - t_{J_1}^{(start)}}{3600} \right\rceil \cdot 3600. \tag{5}$$

   We divide the total time in seconds of using a resource by 3600 in order to convert it to hours, and use the ceiling operator to round this value to complete hours of computation. This equation guarantees a contiguous resource allocation of activities within one hour slot;
3. The next activity not part of the first group $J_{r+1} \notin G_1^{(j)}, r+1 \leq p$, starts after the last hour of computation elapses and the resource is released:

$$t_{J_1}^{(start)} + \left\lceil \frac{t_{J_r}^{(end)} - t_{J_1}^{(start)}}{3600} \right\rceil \cdot 3600 < t_{J_{r+1}}^{(start)}. \tag{6}$$

Successive groups are built until the last activity J_p has been assigned to one group. The second group $G_2^{(j)}$ is constructed in the same way starting from the task J_{r+1} instead of J_1. The same strategy is used for the rest of the groups. Once all the groups have been created, we define the cost $C_{R_j}^{(comp)}$ of using the resource R_j as the number hours required for executing all groups multiplied by the cost per hour:

$$C_{R_j}^{(comp)} = PE_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{A_i \in G_{R_j}^{(k)}} t_{(A_i, R_j)}}{3600} \right\rceil. \tag{7}$$

We compute the economic cost of executing the entire workflow $W = (A, D)$ as the computation cost on all m resources plus the cost for transferring and storing the data:

$$C_W = \sum_{j=1}^{m} C_{R_j}^{(comp)} + \sum_{(A_i, A_j, Data_{ij}) \in D} C_{(A_i, R_j)}^{(data)}. \tag{8}$$

4 Evaluated Techniques

MOHEFT. This method extends HEFT [16] for multi-objective workflow scheduling. MOHEFT [6] requires the number and type of each resource before its execution and the size of the set of tradeoff solutions K. It starts by ranking the tasks in the workflow using the B-rank metric and creating a set S of K empty schedules. Afterwards, MOHEFT iterates over the list of tasks and extends every solution in S by mapping the next task to be executed onto all m possible resources. These new produced schedules are stored in a temporal set S', initially empty. After each iteration, S' replaces S before the next task in the list is considered. Obviously, this strategy results in an exhaustive search if we do not include any restrictions. To avoid it, MOHEFT saves only the best K tradeoffs solutions from the temporary set S' to the set S. These best solutions are selected based on the objective functions and the diversity of the set, i.e., how different are the selected solutions (see [6]). MOHEFT was not originally designed for working with commercial clouds. As a consequence, some modifications are applied in this work to deal with the characteristics of such systems, like the restriction on the maximum number of instances that can be used simultaneously. To deal with this scenario, we extend the algorithm to discard any schedule that use more than N simultaneous resources.

SPEA2*. This algorithm proposed in [17] is a genetic algorithm. It works with a population (set) of solutions which are iteratively recombined with the aim of evolving towards the optima. SPEA2* is initialised with a nearly-optimal solution in terms of makespan (computed using HEFT) and in terms of economic cost (computed with a heuristic aimed at optimising cost). Here, we consider SPEA2* with an population size $K = 10$ and run it for 1000 generations, as performed in [17].

To adapt it for the cloud scenario, we slightly modified SPEA2* in this work. In particular, we enhanced the algorithm with a mechanism for dealing with constrains, similar to the one proposed in [5]. This mechanism always compares first two solutions on the basis of their constraint violation. Solutions which do not violate the constraint are preferred to solutions which violate it (i.e. schedules using 20 or less instances simultaneously are preferred to schedules using more than 20 instances). If both solutions violate the constraint, the one violating it in a lesser extent is preferred (i.e. a solution using 23 machines simultaneously is preferred to a schedule that uses 30). In any other case, solutions are compared considering their makespan and economic cost. By using this mechanism, only the solutions which violate the constraints less survive to the next generation.

It is worth mentioning that SPEA2* is an stochastic algorithm and then it may compute different fronts in different runs of the algorithm. In order to avoid our conclusions be biased by any hazard effect, we run it for five times and always consider the run producing the front with the best values.

5 Experimental Setup

We describe in this section the criteria considered for our comparison, the workflow applications, and the resources infrastructure.

Evaluation Metrics. We consider three criteria for comparing the quality of solutions computed by MOHEFT, and SPEA2*: the shortest makespan of the schedules computed, the cheapest solution reported by each technique. For the sake of comparison we also include HEFT in our graphs describing the results. We also attend to the quality of the computed set of tradeoff solutions. In this last case, we use the hypervolume quality indicator [11]. This indicator assigns a numerical value to each set of tradeoff solutions. The higher this number is, the better the solutions within the set.

We also analyse the computed tradeoff solutions. The idea is to study the balance between makespan and cost, and how much can be gained in one objective by deteriorating the other. For this analysis we rely on a graphical representation of the solutions computed by the two algorithms. The presented graphs start with the most makespan-efficient schedule and continue along the Pareto-front towards the cheapest solution. The two plotted metrics, the cost savings and the makespan deterioration, are presented as percentages relative to the most makespan-efficient solution.

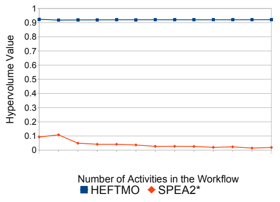
Workflow Applications. We consider in our evaluation two *real-world* workflows coming from our real-world collaborations with domain scientists in the Austrian Grid. These two applications are known as WIEN2k [3] and POV-Ray [13]. The former is a material science workflow for performing structure calculations of solids using density functional theory based on linearized augmented plane-wave; the latter is a free tool for creating three-dimensional graphics. Both workflows contain a high number of independent activities that can be executed in parallel, as well, as sequential parts.

Resource Infrastructure. We consider that the user has access to the default maximum number of $N = 20$ Amazon instances which can be of any of the five types summarised in cite [1]. We assume that no public IP addresses are required for running the experiments on the Amazon EC2 infrastructure. Additionally, the output data transfers from Amazon to the outside Internet are constant, take place only at the end of the workflow execution and thus, do not influence the scheduling results. In this situation, we assume in our experiments that the prices for data sent and received are zero: $PI_{R_i} = 0$ and $PO_{R_i} = 0$.

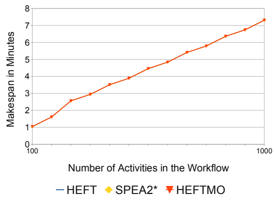
6 Evaluation

In this section we compare MOHEFT and SPEA2*. For the two considered applications, we have evaluated different workflow instances containing between 100 and 1000 activities.

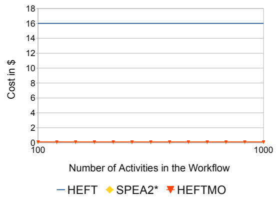
Wien2k. Fig. 1a shows that MOHEFT always outperformed SPEA2* in terms of the hypervolume. The differences in this metric value are high and tend to increase with the number of workflow activities. An explanation for this behaviour is related to the higher difficulty of solving this workflow application. Our hypothesis is that SPEA2* got stuck in some areas of the search space, thus requiring a prohibitively large number of evaluations for increasing the quality of the computed results. The results for makespan and economic cost displayed



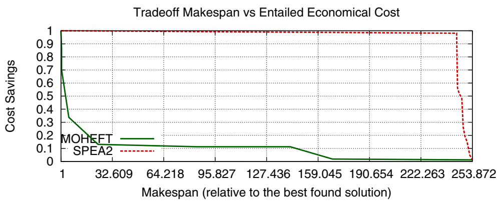
(a) Hypervolume.



(b) Makespan.



(c) Economic cost.



(d) Tradeoff analysis cost vs makespan.

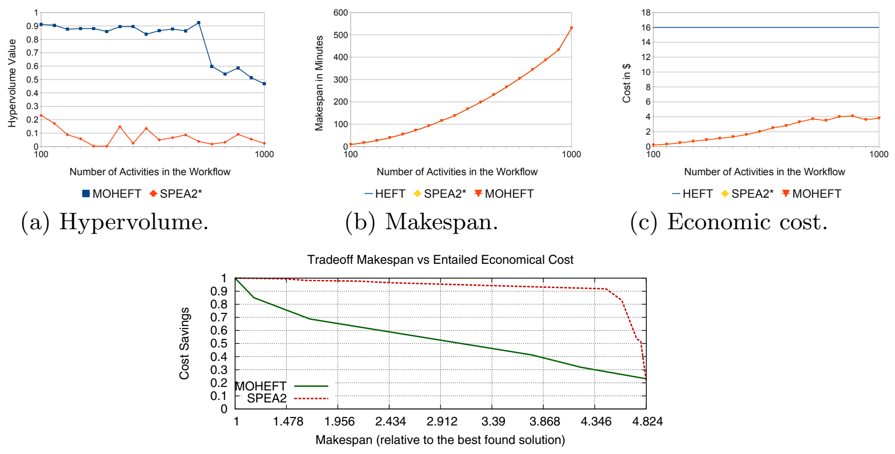
Fig. 1. Evaluation results for the WIEN2k workflow

in Fig. 1b and 1c. In these criteria, both algorithms performed equally well and computed the solution with the lowest makespan and similar cost.

An example comparing the tradeoff solutions delivered by MOHEFT and SPEA2* is illustrated in Fig. 1d. In this case we can see that huge economic savings can be obtained by MOHEFT with only a small increase in makespan. In this case, SPEA2* requires a huge loss in performance for obtaining similar cost results. For example, MOHEFT computed a solution which halves the price of the schedule with the shortest makespan and experienced only a 5% of time deterioration. Meanwhile, a 1% of cost saving in SPEA2* would have required an increase of 250% in makespan. In 23.3% of the cases, SPEA2* computed schedules requiring more than 20 resources, while all solutions computed by MOHEFT met this constraint.

Pov-Ray. Fig. 2a shows the hypervolume of the sets of tradeoff solutions. Also in this case MOHEFT outperformed SPEA2* for all the evaluated workflow sizes. For this application, the higher the number of tasks in the workflow is, the harder is for MOHEFT to compute a set of tradeoff solutions with high quality. This result can be visualized in the hypervolume that decreases with the number of tasks. This behavior is not that obvious for SPEA2*, however, the quality of the computed fronts are of poorer quality than the ones of MOHEFT, as reflected by the low values of the indicator. As in the previous experiment, all the algorithms computed the same solution with the shortest makespan (see Fig. 2b) and the same cheapest schedule too (see Fig. 2c).

An example of the tradeoff solutions computed by MOHEFT and SPEA2* for this application is shown in Fig. 2d. In this case, while MOHEFT found a solution halving the price for the sake of only 3% increase in makespan, SPEA2* required a 4.7% increase. Nevertheless, the small difference between both techniques has



(a) Hypervolume. (b) Makespan. (c) Economic cost.

(d) Tradeoff analysis cost vs makespan.

Fig. 2. Evaluation results for the POV-Ray workflow

to be carefully interpreted, since in 40% of the cases SPEA2* has been unable to produce a workflow schedule using 20 or less resources.

7 Conclusions and Future Work

In this paper we compare the tradeoff solutions computed by two multi-objective workflow scheduler, MOHEFT and SPEA2, in the context of the commercial Clouds Amazon EC2. Both methods are evaluated using real-world applications.

In all experiments, MOHEFT computed schedules with the same makespan as the SPEA2* and similar economic cost; however, MOHEFT outperformed SPEA2* in terms of hypervolume used as an indicator of the quality of the whole set of tradeoff solutions, meaning that for the same cost, MOHEFT computed solutions with shorther makespan. Finally, our experiments revealed that MOHEFT was able to meet the constraints imposed by current commercial Clouds in terms of the maximum amount of instances, while SPEA2* failed on this issue. We also showed the potential of the Pareto front as a tool for decision support in selecting the most appropriate tradeoff solutions. In particular, the visualisation of the Pareto front for some workflow types revealed that one can obtain solutions with a marginal 5% makespan increase by investing half of the money in renting Cloud instances. In future work we intend to evaluate MOHEFT and SPEA2* for other objective functions such as security issues, reliability of spot instances or energy consumption.

Acknowledgements. Juan J. Durillo and Radu Prodan acknowledge Austrian Science Fund project TRP 72-N23 and the Standortagentur Tirol project RainCloud.

References

1. Alexandru, I., Ostermann, S., Yigitbasi, M., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions on Parallel and Distr. Systems 1–16 (2010)
2. Assayad, I., Girault, A., Kalla, H.: A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In: Intern. Conference on Dependable Systems and Networks, DSN 2004. IEEE, Firenze (2003)
3. Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: Wien2k: An augmented plane wave plus local orbitals program for calculating crystal properties. Tech. rep., Institute of Physical and Theoretical Chemistry, TU Vienna (2001)
4. Canon, L.C.: Emmanuel: Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines. Intern. Heterogeneity in Computing (2011)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: Nsga-ii. IEEE Transactions on Evol. Comp. 6, 182–197 (2000)
6. Durillo, J., Fard, H., Prodan, R.: Moheft: A multi-objective lilst-based method for workflow scheduling. In: 4th IEEE Intern. Conference on Cloud Computing Technology and Science (2012)
7. Garg, R., Singh, A.K.: Reference point based multi-objective optimization to workflow grid scheduling. Int. J. Appl. Evol. Comput. 3(1), 80–99 (2012)
8. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility grids: time and cost trade-off management. In: Proceedings of the Thirty-Second Australasian Conference on Computer Science, ACSC 2009, vol. 91, pp. 151–160. Australian Computer Society, Inc., Darlinghurst (2009)
9. Hakem, M., Butelle, F.: Reliability and scheduling on systems subject to failures. In: Proceedings of the 2007 Intern. Conference on Parallel Processing, ICPP 2007. IEEE Computer Society, Washington, DC (2007)
10. Hu, S.Y., Chen, J.F., Chen, T.H.: VON: a scalable peer-to-peer network for virtual environments. IEEE Network 20(4), 22–31 (2006)
11. Knowles, J., Thiele, L., Zitzler, E.: A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. Tech. Rep. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006)
12. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., Albi, E.G.T., Zomaya, A.Y., Tuyttens, D.: A parallel bi-objective hybrid metaheuristic for energy-aware s cheduling for cloud computing systems. Journal of Parallel and Distr. Computing (71), 1497–1508 (2011)
13. Plachetka, T.: POVRAY – Persistence of Vision Parallel Raytracer. In: Proceedings of Computer Graphics Intern. 1998, pp. 123–129 (1998)
14. Ricci, L., Carlini, E.: Distributed virtual environments: From client server to cloud and p2p architectures. In: Smari, W., Zeljkovic, V. (eds.) HPCS, pp. 8–17. IEEE (2012)
15. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in Grid Computing. CoreGrid series. Springer (2007)
16. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distr. Systems 13(3), 260–274 (2002)
17. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM Intern. Conference on Grid Computing, GRID 2007, pp. 10–17. IEEE Computer Society, Washington, DC (2007)