2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

A parallel hybrid evolutionary algorithm for the optimization of broker virtual machines subletting in cloud systems

Santiago Iturriaga, Sergio Nesmachnow
Universidad de la República, Uruguay
{siturria,sergion}@fing.edu.uy

Bernabé Dorronsoro, El-Ghazali Talbi
University of Lille, France
{bernabe.dorronsoro_diaz,El-ghazali.Talbi}@inria.fr

Pascal Bouvry
University of Luxembourg
pascal.bouvry@uni.lu

***Abstract*—This article presents a new parallel hybrid evolutionary algorithm to solve the problem of virtual machines subletting in cloud systems. The problem deals with the efficient allocation of a set of virtual machine requests from customers into available pre-booked resources from a cloud broker, in order to maximize the broker profit. The proposed parallel algorithm uses a distributed subpopulations model, and a Simulated Annealing operator. The experimental evaluation analyzes the profit and makespan results of the proposed methods over a set of problem instances that account for realistic workloads and scenarios using real data from cloud providers. A comparison with greedy heuristics indicates that the proposed method is able to compute solutions with up to 133.8% improvement in the profit values, while accounting for accurate makespan results.**

***Keywords*-parallel evolutionary algorithms; scheduling; cloud computing;**

I. Introduction

Nowadays, cloud computing [1], [2] has emerged as one of the main existing computing paradigms, mainly due to its very interesting features, as elasticity, flexibility, or large computational power, among many others.

Many public and private clouds have appeared in the last years [3]. They all have distinct features, making difficult for users to find the best choice among the existing offers. The *cloud broker* [4] arises as an intermediary between cloud providers and users to help the latter ones in that process. Brokers can simply find the best deals among a set of clouds for the user requirements or even define the best possible design to deploy the user's application in the cloud [4].

This paper focuses on a business model in which the broker sublets on-demand cloud resources to his customers at low prices. The broker owns a set of reserved VMs with different features, and probably from distinct cloud providers, which are offered on-demand to the customers at cheaper prices than those the customer would get from a cloud provider [5]. When the broker does not have enough VMs for executing a customer request without violating the contracted service level agreement, he will buy on-demand VMs in the cloud to satisfy the demand, and the profit of the broker will be reduced (because he will pay the cloud provider more than what he charges to the customer for that VM).

From now, we will refer the reserved VMs of the broker as *reserved instances* (RI) to differentiate from the VMs the customers demand.

The problem of efficiently allocating the customers VM requests into the available RIs arises for the broker. All VMs should be allocated into RIs that are offering at least the same performance requested by the customer, and some quality of service (QoS) levels must be achieved by the solution. This is a resource allocation problem with additional constraints making it more complex. Underutilization of the available RIs must be avoided, as well as the overbooking, which might force the broker reserving on-demand VMs to the cloud provider in order to offer the promised service, despite the money loss. The resource allocation problem itself is NP-hard [6].

The main contributions of this work are: i) the design and implementation of an efficient parallel hybrid evolutionary algorithm to solve the recently proposed virtual machine subletting problem [16], and ii) the evaluation of the proposed method using realistic benchmark instances.

The paper is structured as follows. Next section presents the formulation of the optimization problem tackled. A review of related work on cloud brokering is presented in Section III. Evolutionary computation is briefly introduced in Section IV, just before presenting the hybrid EA in Section V. The experimental evaluation over a set of realistic workloads and scenarios using real data from actual cloud providers is reported in Section VI. Finally, the conclusions and main lines for future work are formulated in Section VII.

II. The Virtual Machine Mapping Problem

The Virtual Machine Mapping Problem (VMMP) in cloud infrastructures considers a set of VMs requested by cloud users to the broker to be executed in the cloud. Each VM is booked on-demand to the broker for a given time and it should start before a specific deadline. Virtual machines have specific hardware demands, that the broker has to fulfill using his own pre-booked VMs, and minimizing the economic cost, thus maximizing his own profit. In case the request of some user(s) cannot be handled with the available RIs, the broker would have to book on-demand VMs in the cloud for them, with the consequent profit reduction.

The VMMP is formalized next. Given the following elements:

- A set of virtual machine requests $VM = \{v_1, \ldots, v_n\}$, each one demanded to execute for a given time $T(v_i)$.
- Each VM has specific hardware demands, including processor speed $P(v_i)$, memory $M(v_i)$, storage $S(v_i)$, and number of cores $nc(v_i)$.
- Virtual machine requests arrive in batches (i.e., hourly, diary). Each VM has an arrival time A_i, according to a stochastic homogeneous Poisson process with parameter (rate) λ.
- The execution of any VM must start before its deadline $D(v_i)$.
- A set of cloud resource instances pre-booked by the broker $B = \{b_1, \ldots, b_m\}$, $m \ll n$, with specific features including processor speed $P(b_j)$, memory $M(b_j)$, and storage $S(b_j)$, according to a predefined list of instance types $t(b_j) \in \{t_1, \ldots, t_k\}$.
- A cost function C for pre-booked cloud resource instances, and a cost function COD for on-demand instances, with $C(b_j) \ll COD(b_j)$. The cost of both functions is given in an hourly basis.
- A pricing function $p(b_j)$ that defines the price the broker charges to the customers per hour for the RI type of b_j. In order to attract customers, the broker should charge for a VM type b_j a lower cost than the on-demand pricing for that kind of VM, i.e., $p(b_j) < COD(b_j)$. Moreover, if the cheapest offered RI that can allocate the VM v_i requested by the user, for instance b_k, is not available, the broker can assign it to another RI of higher capacity, but charging the same amount as for b_k (defined with the best fit function: $BF(v_i)$). This will suppose the revenue to be decreased, but it will prevent the broker from buying (more expensive) on-demand instances, and the customer will be, at the same time, pleased thanks to the better performance offered.

The VMMP in cloud consists in finding a mapping function $f : VM \to B$ for the VM requests $\{v_1, \ldots, v_n\}$ in the available RIs $\{b_1, \ldots, b_m\}$ that maximizes the total broker revenue R, according to the following optimization problem ($ST(v_i)$ states for the starting time of the request v_i, according to the schedule):

$$\max \quad \sum_{j=1}^{j=m} \left(\sum_{i:f(v_i)=b_j} (p(BF(v_i)) - C(b_j)) \times T(v_i) \right) + \sum_{h:ST(v_h)>D(v_h)} (p(BF(v_h)) - COD(BF(v_h))) \times T(v_h)$$

$$\text{subject to} \quad M(v_i) \leq M(b_j),\ P(v_i) \leq P(b_j)$$
$$S(v_i) \leq S(b_j),\ nc(v_i) \leq nc(b_j)$$

where the $BF(v_k)$ function gives the less expensive instance capable of executing the request v_k

In the problem model, deadlines are considered as hard constraints. In case the broker cannot accommodate the VM request to start execution by the specified deadline, he must either use a larger RI offering more resources than those requested (but charging the customer the cost of the requested one) or buy an on-demand instance to fulfill the request. Both solutions obviously accounting for a negative impact in the total cost of the schedule.

The first summation in the revenue objective function accounts for the total profit of the broker thanks to the RIs booked by the customers. The second summation accounts for the additional cost that supposes avoiding the violation of the deadline constraints.

Data transmission for the VMs requests are not considered in the objective function. The model assumes that transmission costs are directly transferred to the user thus the broker cannot take an economic profit from data transmission.

A. *Scheduling approach*

The problem is tackled using a dynamic approach based on *rescheduling*. The scheduling algorithm executes at intervals of a given reschedule time, or when a pre-booked instance is available for a new assignment.

The rescheduling strategy consists in finding a new schedule for executing the incoming requests (in each new batch) and also those requests already submitted that have not finalized yet. Figure 1 graphically describes the process: in time T_R a reschedule is performed, and the new optimization problem considers the new requests arrived plus all the requests that have not yet started at that time, regarding the previously computed schedule. In the new scheduling problem, the calculation of the cost metric must consider the remaining time of those VM requests already in execution at time T_R in each pre-booked instance. To model this situation, at time T_R each pre-booked instance has an available start time $AS(b_i)$.

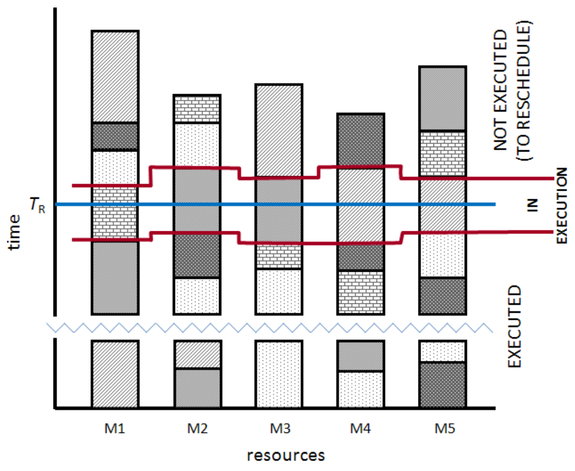

Figure 1. Rescheduling in a dynamic scenario.

III. Literature review

Cloud brokering typically deals [4] with the problems of finding the cloud providers whose offer better suits to the customer needs (both technically and in terms of cost) [7], [8], or providing the customer with the best possible way to deploy his/her application in the cloud [9], [10].

There are in the literature a number of methods for scheduling applications in private resources using cloud bursting technique [11]. These works enhance the local schedulers with the capability of using VMs from the public cloud when additional resources are required. This is a similar concept to the one addressed in this paper, since in the case all reserved VMs are used and a number of users requests cannot start before their deadline, then the broker will buy on-demand instances from the cloud to execute them. However, in our work we do not address the resource provisioning problem, since the broker always work with VMs (either reserved or on-demand) from the public cloud.

Closer to the problem we consider, Wu et al. [12] proposed a mechanism to encourage customers to provide realistic likelihood that they will purchase a given resource, at the reward of price reductions. This mechanism allows the provider to efficiently forecast the required resources, minimizing this way the underutilization and/or overbooking of the available resources, and it will benefit the customer too, who will have the service at a low price. This mechanism was adopted in [5] for the case of a cloud broker subletting reserved VMs to his customers. Then, the broker will use the information given by the customers to decide whether to invest in buying more resources or not, and what kind of resources should be bought. This technique is shown to provide up to 44% increase in the profit of the broker. We investigate in this paper how the broker can optimally manage his VMs for the optimum profit and maximum QoS, allowing the use of on-demand instances to satisfy the needs of users that cannot be satisfied with the current resources, despite the money loss.

IV. Evolutionary computation

This section introduces evolutionary algorithms (EAs) and the parallel hybrid EA proposed in this work.

A. Evolutionary algorithms

EAs are non-deterministic methods that emulate the evolution of species in nature, which have been successfully applied for solving optimization problems underlying many complex real-life applications in the last twenty years [13].

An EA is an iterative technique that applies stochastic operators on a population of *individuals*, which encode tentative solutions of the problem, in order to improve their *fitness*. An evaluation function associates a fitness value to every individual, indicating its suitability to the problem.

The initial population is generated at random or by using a specific heuristic for the problem. Iteratively, the probabilistic application of *recombinations* of individuals or random changes (*mutations*) in their contents, using a selection-of-the-best technique, guides the EA to better solutions.

The stopping criterion usually involves a fixed number of generations or execution time, a quality threshold on the best fitness value, or the detection of a stagnation situation. Specific policies are used for the *selection* of individuals to recombine and to determine which new individuals *replace* the older ones in each new generation. The EA returns the best solution found, regarding the fitness function values.

B. Hybrid EAs

In its broadest sense, hybridization refers to the inclusion of problem-dependent knowledge in a general search algorithm [13]. One possibility is to construct strong hybrid algorithms, where problem knowledge is included as a problem-dependent representation and/or special operators. The other possibility is to combine two or more methods to solve the same problem, constructing weak hybrids and trying to take advantage of their salient features to improve the efficiency or accuracy of the new algorithm. The hybrid algorithm defines a new search pattern which determines when each algorithm is executed, and how the internal states of each algorithm report the results so that the other algorithm can continue. Usually, by exchanging a small set of partial solutions or some statistical values, it is possible to combine algorithms in a (hopefully) efficient manner.

In this work, a weak hybrid algorithm (EA+SA) is designed by combining EA and Simulated Annealing (SA). The EA uses the SA as an evolutionary operator: while the EA provides a good exploration pattern to locate "good" regions of the search space, the SA allows exploitation in the neighborhood of those promising regions.

C. Parallel evolutionary algorithms

Parallel implementations became popular in the last decade as an effort to improve the efficiency of EAs. By splitting the population into several processing elements, parallel evolutionary algorithms (PEAs) allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems [14].

The PEAs proposed in this work are categorized within the *distributed subpopulations* model [15]: the population is divided into several subpopulations (demes) separated from each other. Each deme runs a serial EA, and individuals are only able to interact with other individuals in the deme. An additional operator called *migration* is defined: occasionally some selected individuals are exchanged among demes, introducing a new source of diversity in the EA.

V. A parallel hybrid EA for the VMPP

This section introduces the proposed EA+SA algorithm for tackling the VMPP.

A. Algorithm design and parallel model

EA+SA is a parallel hybrid EA which uses SA as an operator for exploiting promising search space regions.

General description: The schema of EA+SA is presented in Algorithm 1. EA+SA starts by generating an initial population (line 1) by using a randomized Cheapest Instance (rCI) [16] heuristic, which randomly assigns the VM requests to the cheapest RI which is able to fulfill the request requirements. The EA+SA follows the well-known $(\mu + \lambda)$ evolution strategy [17] (lines 4–8), hybridizing a SA operator in order to improve the offspring of the mating operator (line 7). After the population of the deme is evolved, the *migration criterion* is evaluated. If the *migration criterion* evaluates `true`, a set of individuals ν are selected from the current deme population and sent to the next adjacent deme (lines 11–12). In return, a set of solutions ω are received from the previous adjacent deme and combined to the current population (lines 13–14). The evolutionary cycle is executed until the *stopping criterion* is met.

Problem encoding: A fixed-size VM-oriented encoding is used to represent VMMP solutions, allowing an efficient implementation of the evolutionary operators.

Crossover: A special two-point crossover is used. The set of VM requests is randomly split by two cuts producing three subsets. The RI assigned to each request in each of those subsets is exchanged between the two mated parents, scheduling the request in the new RI at the latter feasible time at which it satisfies the request deadline.

Mutation: The mutation operator works as follows. Each VM request ($v \in VM$) in the solution is randomly mutated with a given low probability ($p \leq 0.1$). If v is chosen to be mutated it is rescheduled to be executed by a randomly selected RI ($b \in B$). If the selected RI b fulfills the hardware requirements of the VM request v, a relative position in the scheduling queue of b is randomly selected. If the rescheduled starting time of v satisfies its deadline requirement, then the request is rescheduled. Otherwise, the mutation is discarded.

SA operator: First, the VM request with the worst profit v_{worst} is selected from a subset of randomly chosen VM requests. Then, v_{worst} is rescheduled to execute by an on-demand VM if that improves the profit. Otherwise, a randomly selected subset of RI is explored ($B' \subseteq B$). The $b_{best} \in B'$ RI which improves the most the profit of v_{worst} is selected, and v_{worst} is rescheduled to b_{best} at the latter feasible time at which it satisfies the deadline of v_{worst}.

Parallel model: The parallel model applied in EA+SA arranges the distributed subpopulations using a virtual directed-ring topology. Each subpopulation p_i collaborates with its adjacent neighbors $\{p_{i-1}, p_{i+1}\}$: subpopulation p_i receives candidate solutions from subpopulation p_{i-1}, and sends candidate solutions to subpopulation p_{i+1}.

Algorithm 1 Schema of the distributed EA+SA algorithm.

```
1:  $\mathcal{P} \leftarrow$ generate initial population
2:  while not stopping criterion do
3:      {individual deme evolution}
4:      $\mu \leftarrow$ select parent solutions from $\mathcal{P}$
5:      $\lambda \leftarrow$ mate selected parents in $\mu$
6:      $\tilde{\lambda} \leftarrow$ mutate children in $\lambda$
7:      $\hat{\lambda} \leftarrow$ improve $\tilde{\lambda}$ using SA algorithm
8:      $\mathcal{P} \leftarrow$ select new population from $\{\mu \cup \hat{\lambda}\}$
9:      {collaboration between demes}
10:     if migration criterion then
11:         $\nu \leftarrow$ select solutions to be migrated from $\mathcal{P}$
12:         send $\nu$ to next adjacent deme
13:         $\omega \leftarrow$ receive solutions from previous deme
14:         $\mathcal{P} \leftarrow$ select new population from $\{\mathcal{P} \cup \omega\}$
15:     end if
16: end while
17: return best solution ever found
```

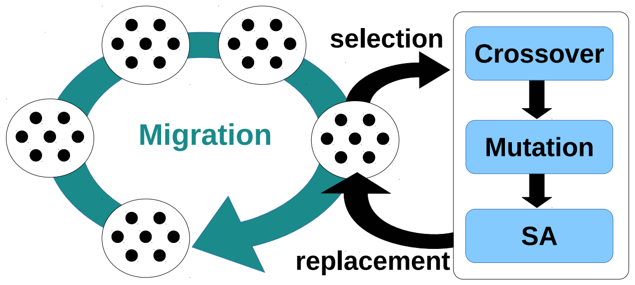

Figure 2. Diagram of the parallel hybrid EA+SA algorithm.

B. Implementation details

The proposed EA+SA algorithm is implemented in C++ using the MALLBA framework [18]. The migration operator is implemented using the MPICH-2 library, a well-known implementation of MPI [19].

VI. Experimental Analysis

This section presents the experimental evaluation of the proposed EA+SA over a realistic set of VMMP intances.

A. Problem instances

We built a set of VMMP instances by following a specific methodology and using real data gathered from public reports, webpages, and nowadays real cloud infrastructures.

The problem instances are defined by: i) a *workload file* with the information about VM requests, including: memory, storage, processor speed, and number of cores requested; and ii) a *scenario file*, with the relevant data for the set of RIs from the broker, including: available memory and storage, processor speed, number of cores, and the cost (both pre-booked and on-demand) and pricing values.

A total number of **100** problem instances are solved in the experimental analysis, by combining workloads and scenarios with diverse characteristics.

Regarding the workloads, we consider batches of 50, 100, 200, and 400 VM requests arriving according to a Poisson process per each scheduling period, each of them with a different duration (from 10 to 200 time units). The considered scenarios built a pre-booked cloud computing infrastructure with 10, 20, and 50 RIs for the broker, by combining VMs from both Amazon and Azure cloud computing services.

For the pricing function, we consider in this work that it is 20% cheaper than the cost on-demand price (i.e. $p(b_j) = 0.8 \times COD(b_j)$). This is a reasonable value for attracting users to the service, while obtaining reasonable profit values.

The VMMP instances are available to download from `http://www.fing.edu.uy/inco/grupos/cecal/hpc/VMMP`.

B. Development and execution platform

The experimental analysis was performed on a 24-Core AMD Opteron Processor 6172 at 2.1GHz, with 24 GB RAM, from *Cluster FING* (`http://www.fing.edu.uy/cluster`).

C. Parameter setting

A fixed stopping criterion of 90 seconds of execution time is used for the EA+SA algorithm evaluation, which is an efficient execution time for on-line cloud planning. 50 independent executions were performed on each VMMP instance, each one using 24 distributed subpopulations.

A configuration analysis was performed using a medium-sized instance in order to find the best values for the crossover (p_c), mutation (p_m), and SA operator (p_{sa}) probabilities. The studied candidate values for each parameter were: $p_c \in \{0.5, 0.7, 0.9\}$, $p_m \in \{0.5, 0.7, 0.9\}$, $p_{sa} \in \{0.1, 0.2, 0.3\}$. A total of 30 independent executions were performed for each of the 27 combination of parameters. Finally, the Friedman Rank Sum (FRS) test was applied on the computed results. A post-hoc analysis of the FRS results showed the most accurate schedules were computed when using $p_c = 0.7$, $p_m = 0.5$, and $p_{sa} = 0.3$. Figure 3 presents the average profit computed by the EA+SA algorithm when using each of the evaluated parameter settings for p_c and p_m.

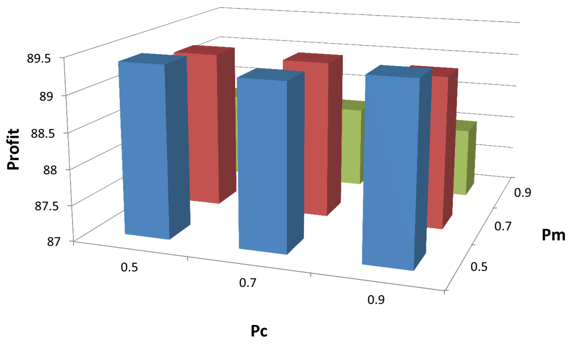

Figure 3. Summary of profit results for p_c and p_m setting analysis.

D. Results and discussion

This subsection summarizes and analyzes the main results of the experimental evaluation of the proposed hybrid EA. The results of the EA+SA algorithm are compared against two different profit-greedy list-scheduling heuristics for the VMMP: Shortest Resource Cheapest Instance (SRCI) and Cheapest Instance (CI) [16].

Table I reports the best and average profit improvement over the results computed by the best known list-scheduling heuristic for the VMMP. Row #1 indicates how many times the EA+SA performed the best (i.e., it is the number one) regarding the profit value. The gap in the makespan value is computed with respect to each heuristic. The makespan metric is defined as the timespan from when the first VM request begins its execution until the last VM request finishes its execution. The makespan gap metric for the EA+SA algorithm is defined as the relative additional makespan time required by the schedules computed by EA+SA when comparing to the makespan of the schedules computed by the CI and SCRI heuristics $\left(gap_{ea+sa} = \frac{makespan_{ea+sa} - makespan_h}{makespan_h}\right)$. Finally, the average relative number of requests violations is reported.

Table I
EA+SA PROFIT IMPROVEMENT AND MAKESPAN GAP OVER THE BEST HEURISTIC, AND AVG. RATIO OF VM REQUEST VIOLATIONS.

dimension	profit improvement			makespan gap		violations
	best	*avg.*	*#1*	*CI*	*SRCI*	*avg.*
50×10	133.8%	43.3%	$^{25}/_{25}$	16.3%	17.7%	2.7%
100×20	47.7%	17.8%	$^{25}/_{25}$	14.1%	10.0%	2.5%
200×20	46.2%	28.7%	$^{25}/_{25}$	8.7%	5.4%	9.7%
400×50	63.7%	26.3%	$^{25}/_{25}$	9.0%	4.5%	5.5%

The results in Table I indicate that EA+SA found the best profit results for all problem instances. In average, the EA+SA algorithm is able to improve the profit computed by the best heuristic in all of the evaluated instances: with average values ranging from 17.8% up to 43.3%. In the best case, the improvement over the best heuristic reached **133.8%** for the smallest instances. Aside that problem dimension, the second best result is obtained for the largest problem instances (namely, 400×50). These extremely good profit values are obtained thanks to the low number of deadline violations in the solutions reported by the algorithm (between 2.7% and 9.7% versus the values reported by the heuristics, ranging from 4.4% to 32.9%). A deadline violation means that the request cannot be performed in any of the RIs owned by the broker on time, implying that he has to buy an on-demand VM to the cloud provider to perform it within the stipulated deadline.

In terms of makespan, we can see in Table I that EA+SA provides slightly worse results than the heuristics (between 4.5% and 17.7%); the lowest values are obtained or the largest instances. The reason is that the EA+SA algorithm has more requests to schedule in the available RIs, because the percentage of violated requests is lower. This issue obviously directly impacts on the makespan.

We also investigated the benefits of the parallel model to compute more accurate solutions when additional computing resources are available. Table II presents the average profit improvement (with respect to the best compared heuristic in every case) computed by the EA+SA algorithm when using 1, 8, and 24 distributed demes.

Table II
AVERAGE PROFIT IMPROVEMENT OF EA+SA OVER THE BEST HEURISTIC VARYING THE AMOUNT OF DISTRIBUTED DEMES.

dimension	**average profit improvement**		
	1 deme	*8 demes*	*24 demes*
50×10	42.89±0.39%	43.20±0.16%	43.29±0.09%
100×20	17.09±0.52%	17.60±0.31%	17.80±0.20%
200×20	24.83±1.36%	27.57±0.91%	28.71±0.75%
400×50	21.34±2.04%	24.57±1.44%	26.30±1.23%

The experimental analysis shows that increasing the number of demes of the EA+SA algorithms, and therefore the number of evaluations performed, allows to improve the accuracy of the algorithm, enhancing the average profit. This accuracy improvement increases with the dimension of the problem instances. The improvement for the 24 demes algorithm with respect to the one using 1 deme ranges from 0.93% for the smallest instances to 23.24% for the largest ones. Results also show that the greater the number of demes, the more robust the EA+SA algorithm is, presenting a lower standard deviation of the computed profit.

VII. CONCLUSIONS AND FUTURE WORK

This article presents a novel parallel hybrid EA to solve the problem of virtual machines mapping, which arises for the cloud broker that sublets reserved instances as on-demand ones to his customers at lower prices than those offered by public cloud providers (we consider 20% cheaper prices in this work). The problem was recently modeled in [16].

The new proposed algorithm is shown to clearly outperform the best existing results in the literature [16] in an affordable amount of time. The profit of the broker is increased by up to 133.8% when using the proposed technique, which only requires 90 seconds of execution time. Additional scalability tests showed that the profit improves when increasing the computational effort (by using more cores in parallel), particularly for the biggest problem instances.

The main lines for future work include to further analyze the behavior and dynamics of the new technique, as well as to investigate on other more accurate methods. Designing an accurate forecasting technique to predict the resources the broker will need in the future is another important line of future research.

REFERENCES

[1] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing: Principles and Paradigms*. Wiley, 2011.

[2] I. Foster, Y. Zhao, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop*, 2008.

[3] B. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Fifth International Joint Conference on INC, IMS and IDC (NCM)*, 2009, pp. 44–51.

[4] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: Taxonomy and survey," *Software: Practice and Experience*, pp. 1–22, online first, 2012.

[5] O. Rogers and D. Cliff, "A financial brokerage model for cloud computing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, no. 2, pp. 1–12, 2012.

[6] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. of the 16th international conference on World Wide Web*. ACM, 2007, pp. 331–340.

[7] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. of the 10th International Conference on Algorithms and Architectures for Parallel Processing*. Springer-Verlag, 2010, pp. 13–31.

[8] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, 2012.

[9] U. Lampe, "Optimizing the distribution of software services in infrastructure clouds," in *IEEE World Congress on Services*, 2011, pp. 69–72.

[10] F. Legillon, N. Melab, D. Renard, and E.-G. Talbi, "Cost minimization of service deployment in a multi-cloud environment," in *IEEE International Conference on Evolutionary Computation*, 2013, p. to appear.

[11] R. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds," in *Proc. of the 13th International Conference on Web Information System Engineering*, 2012, pp. 28–30.

[12] F. Wu, L. Zhang, and B. Huberman, "Truth-telling reservations," *Algorithmica*, vol. 52, no. 1, pp. 65–79, 2008.

[13] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*. Oxford University Press, 1997.

[14] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[15] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, 2002.

[16] S. Nesmachnow, S. Iturriaga, B. Dorronsoro, E.-G. Talbi, and P. Bouvry, "List scheduling heuristics for virtual machine mapping in cloud systems," in *VI High Performance Computing Latin America Symposium*, Mendoza, Argentina, 2013.

[17] S. Sivanandam and S. Deepa, *Introduction to Genetic Algorithms*, 1st ed. Springer-Verlag, 2007.

[18] E. Alba, G. Luque, J. Garcia-Nieto, and G. Ordonez, "MALLBA: a software library to design efficient optimisation algorithms," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 74–85, 2007.

[19] W. Gropp, "MPICH2: A new start for MPI implementations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2002, pp. 7–7.