INFORMATICA, 2015, Vol. 26, No. 1, 111–134

DOI: http://dx.doi.org/10.15388/Informatica.2015.41

Metaheuristics for the Virtual Machine Mapping Problem in Clouds

Sergio NESMACHNOW[1]*, Santiago ITURRIAGA[1],
Bernabé DORRONSORO[2], El-Ghazali TALBI[3], Pascal BOUVRY[4]
[1]*Centro de Cálculo, Universidad de la República, Uruguay*
[2]*Department of Computer Science, University of Cádiz, Spain*
[3]*LIFL, University of Lille 1, France*
[4]*FTCS, University of Luxembourg, Luxembourg*
e-mail: sergion@fing.edu.uy, siturria@fing.edu.uy, bernabe.dorronsoro@uca.es, el-ghazali.talbi@lifl.fr, pascal.bouvry@uni.lu

Received: January 2014; accepted: June 2014

Abstract. This article presents sequential and parallel metaheuristics to solve the virtual machines subletting problem in cloud systems, which deals with allocating virtual machine requests into pre-booked resources from a cloud broker, maximizing the broker profit. Three metaheuristic are studied: Simulated Annealing, Genetic Algorithm, and hybrid Evolutionary Algorithm. The experimental evaluation over instances accounting for workloads and scenarios using real data from cloud providers, indicates that the parallel hybrid Evolutionary Algorithm is the best method to solve the problem, computing solutions with up to 368.9% profit improvement over greedy heuristics results while accounting for accurate makespan and flowtime values.

Key words: cloud computing, planning, brokering.

1. Introduction

In the last years, cloud computing (Buyya *et al.*, 2011; Foster *et al.*, 2008) emerged as one of the main existing computing paradigms, mainly due to several very interesting features it provides, including elasticity, flexibility, or large computational power, among many others. Cloud computing has raised the interest of both academic and industrial research communities, by providing a computing model which is able to cope efficiently with complex problems involving hard computing functions and handling very large volumes of data.

Cloud computing provides a stack composed of different kinds of services to users, including: infrastructure as a Service (IaaS), dealing with resources as servers, storage, or networks; Platform as a Service (PaaS), which provide an operating system as well as a set of tools and services to the user; or Software as a Service (SaaS) that allows providers to grant customers with access to licensed software.

Many different public and private clouds have appeared in the last years (Rimal *et al.*, 2009). They all have distinct features, making difficult for users to find the best choices

*Corresponding author.

among the many existing offers. In this model, the *cloud broker* (Grozev and Buyya, 2012) 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 users' applications in the cloud (Grozev and Buyya, 2012).

This paper focuses on a IaaS-level business model for cloud computing, in which the broker sublets on-demand cloud resources to his customers at low prices. The broker is considered here as a virtual cloud. It owns a set of reserved Virtual Machines (VMs) with different features, and probably from distinct cloud providers, which are offered to the customers as on-demand resources at cheaper prices than those the customer would get from a cloud provider (Rogers and Cliff, 2012).

This business model is feasible and profitable due to the large price difference between reserved and on-demand VMs in the cloud (Rogers and Cliff, 2012). Additionally, in the case the broker cannot find an available RI to deploy a customer VM request without violating the contracted service level agreement, he will buy on-demand VMs in the cloud to satisfy the demand, with the consequent profit reduction (because he will pay the cloud provider more than what he charges to the customer for that VM). This approach is technically possible, as the technology required is similar to that of the *cloud bursting* technique used in hybrid clouds (Mattess *et al.*, 2011).

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

The considered model does not only benefit customers, but it also could benefit cloud providers, who have more information (from the RIs by the broker) to be able to forecast the resources that will be needed in the future.

The optimization problem of efficiently allocating the customers' VM requests into the available RIs arises for the broker. All VMs must be allocated into RIs that are offering at least the same performance requested by the customer, in order to satisfy the arranged service level agreement (SLA) for every VM. The general form of the resource allocation problem is NP-hard (Tang *et al.*, 2007), since it is a generalization of the multidimensional knapsack problem (Kellerer *et al.*, 2004), and the problem variant tackled in this article is even more complex due to the additional constraints included to satisfy the SLA for each assignment. Underuse of the available RIs (meaning that they are either not sublet, or sublet to a customer requiring a VM of less capacity) must be avoided, because it will cause a reduced revenue for the broker. In addition, overbooking should be avoided too, because it might force the broker to reserve on-demand VMs to the cloud provider in order to offer the promised service, despite the money loss.

In this line of work, the main contributions of the research reported in this article are: (i) the formalization of the virtual machine mapping problem in clouds, introduced in Iturriaga *et al.* (2013), Nesmachnow *et al.* (2013) (this paper extends those works); (ii) the design and implementation of efficient sequential/parallel metaheuristics to solve the problem, and ii) the evaluation of the proposed methods using a comprehensive set of realistic benchmark instances, built by gathering data from real cloud providers.

The rest of the paper is structured as follows. Next section presents the formulation of the optimization problem tackled in this work. A review of related work on cloud bro-

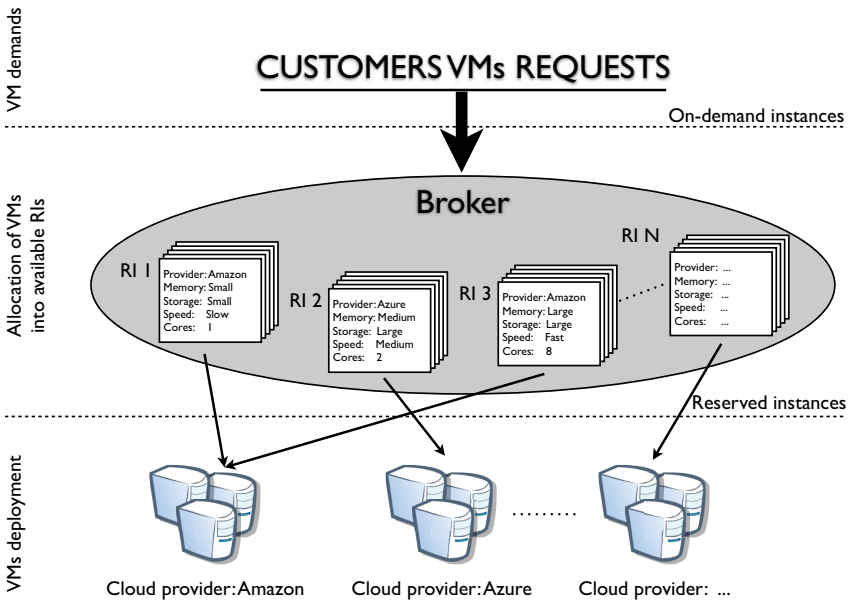


Fig. 1. The problem model: the broker owns a number of reserved instances in different cloud providers to which the on-demand VM requests from its customers are allocated.

kering is presented in Section 3. Metaheuristic algorithms and parallel metaheuristics are briefly introduced in Section 4, just before presenting the Simulated Annealing, Genetic Algorithm and the hybrid Evolutionary Algorithm proposed to tackle the problem in Section 5. The experimental evaluation of the studied resolution techniques over a set of realistic workloads and scenarios using real data from actual cloud providers is reported in Section 6. Finally, the conclusions and main lines for future work are formulated in Section 7.

2. 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. VMs also have specific hardware demands, that the broker has to fulfill using his own pre-booked VMs—that we call RIs—, and minimizing the economic cost, thus maximizing his own profit. Figure 1 presents a graphical description of the problem model considered in this article.

When some user(s) request(s) cannot be handled using the available RIs, the broker may decide whether rejecting the requests or booking on-demand VMs in the cloud for executing them. In our approach, the broker follows the second option. Our argument is that even when the broker action implies a consequent profit reduction, we consider that the reputation of the broker is of major importance to attract new customers and

prevent existing ones from leaving. Therefore, we assume that it is a better strategy keeping reputation to the maximum, at the cost of loosing some money in those cases. Additionally, this consideration provides a lower bound for the broker profit, which might be increased by rejecting the requests causing the overload.

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 that must be respected, 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, daily). Each VM request has an arrival time A_i, according to a stochastic homogeneous Poisson process with rate λ.
- The execution of any VM must start before its deadline $D(v_i)$.
- A set of cloud reserved instances the broker owns $B = \{b_1, \ldots, b_m\}$, $m \ll n$, with specific features including processor speed $P(b_j)$ and memory $M(b_j)$ and storage $S(b_j)$ capacities, according to a predefined list of instance types $t(b_j) \in \{t_1, \ldots, t_k\}$.
- A cost function C for cloud RIs, 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. The number and kind of RIs to book is a decision to be made by the broker. This problem is out of scope of this paper and it will be tackled in our future work.
- A pricing function $p(b_j)$ that defines the (hourly) price the broker charges the customers for the RI type b_j. In order to attract customers, the broker should charge for a RI type b_j a lower cost than the on-demand price for that kind of instance, i.e., $p(b_j) < COD(b_j)$. Moreover, if the cheapest RI that can allocate a VM v_i requested by the user (i.e., type b_k, defined with the *best fit* function $BF(v_i)$) is not available, the broker can assign v_i to a higher capacity RI, but charging the same amount as for b_k. This could imply the revenue to be decreased, but it will prevent the broker from buying a (more expensive) on-demand instance, and the customer will be, at the same time, pleased thanks to the better performance offered.

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

$$\text{subject to } M(v_i) \leqslant M(b_j), \quad P(v_i) \leqslant P(b_j), \tag{2}$$

$$S(v_i) \leqslant S(b_j), \quad nc(v_i) \leqslant nc(b_j,) \tag{3}$$

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

The VMMP in cloud platforms consists in finding a *mapping function* $f : VM \rightarrow B$ for the VM requests $\{v_1, \ldots, v_n\}$ into the available RIs $\{b_1, \ldots, b_m\}$ that maximizes the total

broker revenue, according to the optimization problem formulated in Eq. (1), where $ST(v_i)$ stands for the starting time of the VM request v_i, regarding the computed assignment.

In the problem model, deadlines are considered as hard constraints. In case the broker cannot accommodate a given VM request to start execution by the specified deadline, he must either use a higher capacity 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 alternatives obviously account for a negative impact for the broker in the total cost of the mapping but, at the same time, they contribute to keep/improve the broker reputation.

The first summation in the revenue objective function accounts for the total profit of the broker, corresponding to those VM requests executing in a suitable RI booked by the customers. The second summation accounts for the additional cost (corresponding to on-demand instances in the public cloud) that supposes avoiding the violation of the deadline constraints.

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

3. Literature Review

Despite the large number of papers existing in the literature about cloud brokering, only very few of them tackle similar problems as the one addressed in this article.

Cloud brokering (Grozev and Buyya, 2012) typically deals with the problems of finding the cloud providers whose offers better suit to a set of customer needs (both technically and in terms of cost) (Buyya *et al.*, 2010; Sotiriadis *et al.*, 2011, 2012; Tordsson *et al.*, 2012), or providing the customer with the best possible way to deploy his/her applications in the cloud (Lampe, 2011; Legillon *et al.*, 2013). Therefore, cloud brokering does not deal with the deployment of VMs in the servers of the cloud (Besis *et al.*, 2014; Buyya *et al.*, 2011).

There are in the literature a number of methods for scheduling applications in private resources using the cloud bursting technique (Calheiros and Buyya, 2012). These proposals are based on enhancing 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 our article, since in the case all RIs are in use and a number of users requests cannot start before their deadline, then the broker will buy on-demand instances from the cloud to execute those requests. 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.* (2008) 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 lower price.

The previous approach was adopted in Rogers and Cliff (2012) 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.

In this article, we investigate 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.

4. Metaheuristics

This section describes metaheuristic techniques for optimization and the algorithms applied in this work to solve the VMMP.

4.1. *Metaheuristics*

Nowadays, many problems arising in real-world applications are intrinsically complex. In practice, many optimization problems are NP-hard, thus traditional exact techniques (backtracking, dynamic programming, enumeration algorithms, etc.), are not useful for solving them, as they demand very large execution times when solving realistic problem instances. Metaheuristics are soft computing methods that allow computing sub-optimal (even optimal) solutions, for hard-to-solve problems in reasonable execution times (Blum and Roli, 2003; Talbi, 2009; Nesmachnow, 2014).

Several classification criteria have been proposed for metaheuristics in the related literature. One of the most used classifications take into account the number of tentative solutions handled, and two categories are recognized: *trajectory-based* and *population-based*. Trajectory-based metaheuristics work with a single solution, which is iteratively modified in each step, to be replaced by another (often the best) solution found in its neighborhood. Population-based metaheuristics work with a set of multiple candidate solutions, which are modified and/or combined following some common guidelines, and some of them are replaced by newly generated solutions (often by the best ones).

In this article we apply both sequential and parallel versions for a trajectory-based metaheuristic (Simulated Annealing) and two population-based methods (Evolutionary Algorithm and an Hybrid Algorithm) to solve VMMP. The main features of the metaheuristic methods applied in this work are described in the following subsections.

4.2. *Simulated Annealing*

Simulated Annealing (SA) is a metaheuristic technique inspired on the annealing process of metals. It was the first metaheuristic proposed (thirty years ago), although

the term "metaheuristic" was not yet defined by that time (Kirkpatrick *et al.*, 1983; Černý, 1985). SA is a local search optimization method based on a Metropolis-Hastings (Markov chain/Monte Carlo) algorithm to find the lowest energy (most stable) orientation for an n-body system. By applying such analogy, SA defines a generalization of the Monte Carlo approach to solve combinatorial optimization problems.

SA maintains a current solution for the problem (analogous to the current state of a physical system) with an associated objective function (analogous to the energy function) whose global minimum (analogous to the ground state) is searched. SA employs a temperature T to control the probability of accepting/moving to a solution that does not improve the objective function value. The rationale behind this decision is to try to escape from local optima. There is no obvious physical analogy for the temperature T (as such a free parameter does not exist in the combinatorial optimization problem), and so defining an appropriate annealing schema for avoiding local optima is usually a crafty task.

The parameters for the SA method, which are often empirically determined, are the initial temperature, the number of iterations performed at each step (the Markov chain length), and the temperature decreasing schema.

SA is a trajectory-oriented technique: it maintains only one sub-optimal solution for the problem and explores the search space via certain local search transition operators. Using such operators, it is possible to explore multiple points in the neighborhood of the current solution when solving a particular problem.

4.3. *Evolutionary Algorithms*

EAs are non-deterministic methods that simulate 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 (Bäck *et al.*, 1997).

An EA is an iterative technique (each iteration is called a *generation*) that applies stochastic operators on a population of *individuals*, which encode tentative solutions of the problem. The objective is to improve their *fitness*, a measure related to the objective function. 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 *variation operators* like the *recombinations* of individuals or random changes (*mutations*) in their contents, are guided by a selection-of-the-best technique to tentative solutions of higher quality.

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.

Genetic Algorithms (GA) (Goldberg, 1989) are the most popular variant for an EA. A GA uses the recombination or *crossover* as the main operator in the search, and the mutation is used as a secondary operator, applied with a very low probability in order to provide diversity to the population.

4.4. *Hybrid EAs*

In its broadest sense, hybridization refers to the inclusion of problem-dependent knowledge in a general search algorithm (Bäck *et al.*, 1997).

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 by determining when each component is executed, and how the internal states of each component report the results to the other component. Usually, by exchanging a small set of partial solutions or some statistical values, it is possible to combine algorithms in an efficient manner (Talbi, 2002).

In this article, we designed the hybrid EA+SA algorithm by combining EA and SA, following a weak hybridization approach: the EA uses the SA as an inner variation operator. The main idea is that the EA provides an explorative behavior to locate promising regions of the search space, and the SA operator allows exploiting and improving accurate solutions found, by searching in the neighborhood of (already found) good solutions.

We designed the hybrid EA+SA algorithm after observing in initial experiments that both standard methods, SA and GA, were able to find accurate solutions for different problem dimensions. Thus, we aim at developing a more accurate optimization technique for the VMMP, by combining the main features of SA and GA.

4.5. *Parallel Metaheuristics*

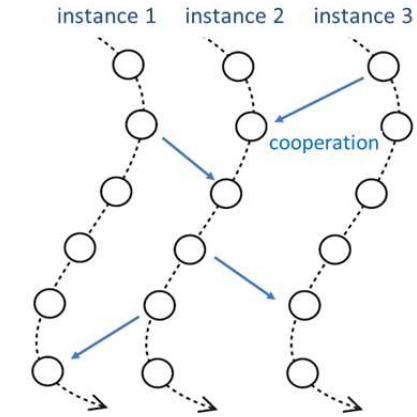
Parallel models for metaheuristics have been conceived to enhance and speed up the search (Alba *et al.*, 2013). By splitting the search into several processing elements, parallel metaheuristics allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems.

The main models for parallelizing trajectory-based metaheuristics are:

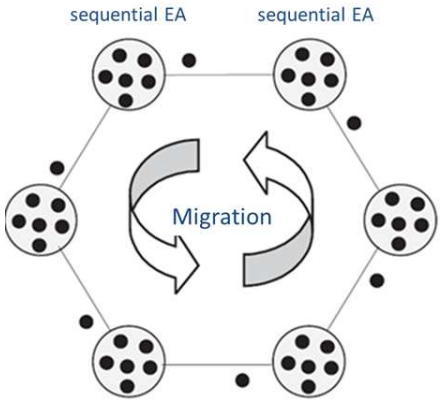
- *Parallel multi-start model*: launches several instances of the method, which may be heterogeneous/homogeneous, independent/cooperative, start from the same or different solution(s), and configured with the same or different parameters.
- *Parallel moves model*: distributes the current solution to several processes, each one explores by applying a search based on its candidate solution and the results are returned to the main process.
- *Move acceleration model*: evaluates several moves in a parallel centralized way, by parallelizing the objective function using an aggregation approach.

The main models for parallelizing population-based metaheuristics take into account the criterion used to organize the population:

- *Master-slave model*: follows a functional decomposition using a hierarchic structure: a master process performs the evolutionary search, and controls a group of slave



(a) Parallel multi-start with cooperation



(b) Distributed subpopulations

Fig. 2. Parallel models for metaheuristics applied to solve the VMMP.

processes that evaluate the objective function, which often requires larger computing time than the search operators.

- *Distributed subpopulations model*: splits the population in subpopulations, each one running a sequential metaheuristic. The solutions only can interact within their subpopulation. A migration operator exchanges some selected solutions among subpopulations, introducing a new source of diversity.
- *Cellular model*: uses an underlying spatial structure for the population, usually a two-dimensional grid. Interactions are restricted to neighboring solutions, and characteristics gradually propagate through the grid. This feature is useful to provide diversity in the population, often improving the search.

The strategies applied for the parallel metaheuristic algorithms implemented in this work to solve VMMP are: (i) a parallel multi-start with cooperation model for SA (Fig. 2(a)), and (ii) a distributed subpopulations model for the GA and the hybrid EA+SA (Fig. 2(b)).

5. Metaheuristics for VMMP

This section introduces the proposed metaheuristic algorithms for tackling VMMP.

5.1. *Simulated Annealing*

The SA algorithm is based on a movement operator that works over random VM requests that generate low profit for the broker, according to the schema presented in Algorithm 1.

The SA algorithm starts by generating an initial solution using the randomized Cheapest Instance (rCI) heuristic (Nesmachnow *et al.*, 2013), which randomly assigns the VM requests to the cheapest RI which is able to fulfill the request requirements (line 1). After that, the cycle that performs the search is started. First, a subset μ (cardinality $\frac{n}{10}$) of VM requests is randomly chosen to perform the search (line 3), and the VM request with the worst profit v_W is selected from that subset (line 4). Then, the movement operator in the SA algorithm tries to reschedule v_W to execute by an on-demand VM, in case this choice improves the broker profit (line 5). Otherwise, a randomly selected subset $B' \subseteq B$ (cardinality $\frac{m}{2}$) of RI is explored (line 7). The $b_{best} \in B'$ RI which improves the most the profit of v_{worst} is selected (line 8), and v_W is rescheduled to b_{best} at the latter feasible time at which it satisfies the deadline of v_W (line 9). Only in the parallel version, the cooperation takes place by exchanging solutions when the collaboration criterion is met (lines 12–18).

The searching cycle is repeated until the stopping criterion is met.

Algorithm 1 Sequential/parallel SA for VMMP.

```
1:  s ← generate initial solution using rCI
2:  while not stopping criterion do
        {apply movement}
3:      Ω ← select μ random VM requests
4:      v_W ← worst profit VM request in Ω
5:      s̃ ← reschedule v_W to on-demand VM on s
6:      if profit(s̃) < profit(s) then
7:          B̃ ← select subset in B
8:          b_best ← RI ∈ B' which improves the most the profit of v_W
9:          s̃ ← reschedule v_W to b_best in s (at the latter feasible time satisfying v_W deadline)
10:     end if
11:     s ← s̃
        {only in parallel SA: collaboration}
12:     if collaboration criterion then
13:         send s to adjacent SA
14:         receive ŝ from adjacent SA
15:         if profit(ŝ) > profit(s) then
16:             s ← ŝ
17:         end if
18:     end if
19: end while
20: return s
```

Algorithm 2 Sequential/parallel EA for VMMP.

1: $\mathcal{P} \leftarrow$ generate initial population
2: **while** not stopping criterion **do**
   {individual deme evolution}
3: $\Omega \leftarrow$ select μ parent solutions from $\mathcal{P}$
4: $\Phi \leftarrow$ mate selected parents in ω, generate λ offspring
5: $\widehat{\Phi} \leftarrow$ mutate children in Φ
   {only in sequential/parallel EA+SA}
6: $\widehat{\Phi} \leftarrow$ improve $s \in \widetilde{\Phi}$ using SA algorithm
7: $\mathcal{P} \leftarrow$ select new population from $\{\mu \cup \widehat{\Phi}\}$
   {only in parallel EA: deme collaboration}
8: **if** migration criterion **then**
9: $\nu \leftarrow$ select solutions to be migrated from $\mathcal{P}$
10: send ν to next adjacent deme
11: $\omega \leftarrow$ receive solutions from previous deme
12: $\mathcal{P} \leftarrow$ select new population from $\{\mathcal{P} \cup \omega\}$
13: **end if**
14: **end while**
15: **return** best solution ever found

5.2. *Genetic Algorithm*

The proposed GA follows the schema presented in Algorithm 2 for an EA applied to the VMMP. We refer to it generically as "schema for an EA", because the same general schema is applied in both GA and EA+SA, but the latter method incorporates a SA-based operator in the search.

The GA algorithm starts by generating an initial population (line 1) using a randomized Cheapest Instance (rCI) heuristic. rCI randomly assigns the VM requests to the cheapest RI which is able to fulfill the request requirements (Nesmachnow *et al.*, 2013).

Problem encoding. Each individual in the population is represented in memory using a fixed-size VM-oriented encoding, which allows an efficient implementation of the variation operators.

Crossover. A special version of the well known two-point crossover (Goldberg, 1989) is used in the proposed GA. The set of VM requests is randomly split by using two cutting points, thus producing three subsets. The RI assigned to each request in each of those subsets is exchanged between the two mated parents. When exchanging the RIs of a VM request between parents, the request is scheduled in the destination RI at the latter feasible time at which it satisfies its deadline.

Mutation. The mutation operator works as follows. When mutating an individual in the population, each VM request ($v \in VM$) in the solution is randomly changed with a given low probability ($p \leqslant 0.1$). 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, if the selected rescheduling is not feasible, the mutation is discarded.

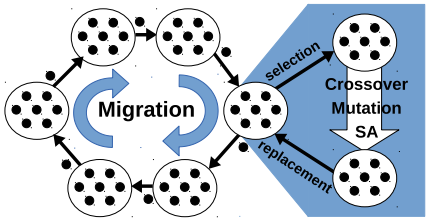


Fig. 3. Diagram of the parallel hybrid EA+SA algorithm.

Only in the parallel version of GA, the migration operator is applied between the different demes (or subpopulations) when the migration criterion is met (lines 8–13). A detailed description of the migration operator, which is the same for both GA and EA+SA, is provided in the following subsection.

5.3. *The Hybrid EA+SA Algorithm*

EA+SA is a hybrid EA that incorporates an SA method as an inner operator for exploiting solutions found in promising regions of the search space. The EA+SA algorithm follows the general schema described in Algorithm 2, hybridizing a SA operator in order to improve the offspring produced by the mating operator (line 6 in Algorithm 2).

The problem encoding, initialization, crossover and mutation operators applied in EA+SA are the same as those used in the standard GA, which were described in the previous subsection.

In the parallel version of EA+SA, the *migration criterion* is evaluated after the population of the deme is evolved. If the *migration criterion* evaluates `true`, a set of individuals v are selected from the current deme population and sent to the next adjacent deme (lines 9–10). In return, a set of solutions ω are received from the previous adjacent deme and combined to the current population (lines 11–12). The evolutionary cycle is executed until the *stopping criterion* is met.

Parallel model. The parallel model applied in the EA+SA algorithm arranges the distributed subpopulations using a virtual directed-ring topology. Each subpopulation p_i ($i = 1, \ldots, p$) collaborates during its evolution with its adjacent neighbors $\{p_{i-1}, p_{i+1}\}$. This collaboration is arranged in a unidirected ring, such that subpopulation p_i receives candidate solutions from subpopulation p_{i-1}, and sends candidate solutions to subpopulation p_{i+1}.

Figure 3 presents a diagram for the distributed subpopulations parallel model used in the EA+SA algorithm.

5.4. *Implementation Details*

The three proposed metaheuristic algorithms to solve VMMP are implemented in C++ language using the MALLBA framework (Alba *et al.*, 2007), and compiled with GNU g++. The migration operator in the parallel versions is implemented using the MPICH-2

library, a well-known implementation of the MPI library for parallel and distributed programming (Gropp, 2002).

6. Experimental Analysis

This section presents the experimental evaluation of the proposed metaheuristic algorithms to solve VMMP over a realistic set of problem instances.

6.1. *Problem Instances*

For the experimental analysis of the proposed methods, 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 two data sets: (i) a *workload file* storing 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, the pre-booked cost, the on-demand cost, and the pricing values for customers. A total number of **400** problem instances are solved in the experimental analysis, by combining workload and scenario files with diverse characteristics and dimensions.

Regarding the workloads, we consider batches of 50, 100, 200, and 400 VM requests arriving according to a Poisson process, each of them with a different duration (from 10 to 200 time units).

The considered scenarios build pre-booked cloud computing infrastructures with 10, 20, 30, and 50 RIs for the broker, by combining VMs from both Amazon and Azure cloud computing services, according to the real details and data presented in Table 1 (price and costs are in US dollars, all configuration and prices are updated as for May, 2013).

The VM configurations selected account for different configurations, including small and average machines (instances type #1, #2, and #3), large machines (instances type #4 and #6) and instances with large memory, CPU and/or storage (instances type #5, #7 and #8).

Table 1
VMs types considered to built the broker pre-booked cloud infrastructure.

#	VM id	Provider	Memory	Storage	Processor	nc	Price	C	COD
1	m1.small	Amazon	1.7 GB	160 GB	1.0 GHz	1	0.048	0.027	0.06
2	m1.medium	Amazon	3.75 GB	410 GB	2.0 GHz	2	0.096	0.054	0.12
3	A2.medium	Azure	3.5 GB	489 GB	1.6 GHz	2	0.096	0.09	0.12
4	m1.large	Amazon	7.5 GB	850 GB	2.0 GHz	4	0.192	0.108	0.24
5	m2.xlarge	Amazon	17.1 GB	420 GB	3.25 GHz	2	0.192	0.136	0.24
6	A3.large	Azure	7.0 GB	999 GB	1.6 GHz	4	0.328	0.18	0.41
7	c1.xlarge	Amazon	7.0 GB	1690 GB	2.5 GHz	8	0.384	0.316	0.48
8	A4.xlarge	Azure	14.0 GB	2039 GB	1.6 GHz	8	0.464	0.36	0.58

Regarding the pricing function, we consider in this article that it is 20% cheaper than the on-demand cost value (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 for the broker.

All the 400 VMMP instances used in this article to evaluate the proposed sequential/parallel metaheuristics are publicly available for downloading from the VMMP website `http://www.fing.edu.uy/inco/grupos/cecal/hpc/VMMP`.

6.2. *Experimental Platform*

The experimental analysis was performed on a 24-Core AMD Opteron Magny-Cours Processor 6172 at 2.1 GHz, with 24 GB RAM and CentOS Linux, from the HPC facility *Cluster FING*, Universidad de la República, Uruguay (Cluster FING website: `http://www.fing.edu.uy/cluster`.

6.3. *Parameter Setting*

A stopping criterion of 90 seconds of execution time is used for all the metaheuristics studied in this article. The proposed stopping criterion is an efficient execution time for on-line cloud planning, and it is in accordance with related works on grid/cloud planning in the literature (Nesmachnow *et al.*, 2010, 2012; Xhafa *et al.*, 2008).

A configuration analysis was performed using two medium-sized VMMP instances in order to find the best values for the crossover probability (p_C) and mutation probability (p_M) in the GA, and the SA operator probability (p_{SA}) in the EA+SA metaheuristic.

The studied candidate values for each parameter were: $p_C \in \{0.5, 0.7, 0.9\}$, $p_M \in \{0.5, 0.7, 0.9\}$, $p_{SA} \in \{0.1, 0.2, 0.3\}$. The SA algorithm was always executed using an exponential temperature decay schema, with exponent 0.99, and a Markov chain length of 5.

A total number of 30 independent executions were performed for each of the 27 parameters combinations. 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 $p_{SA} = 0.3$.

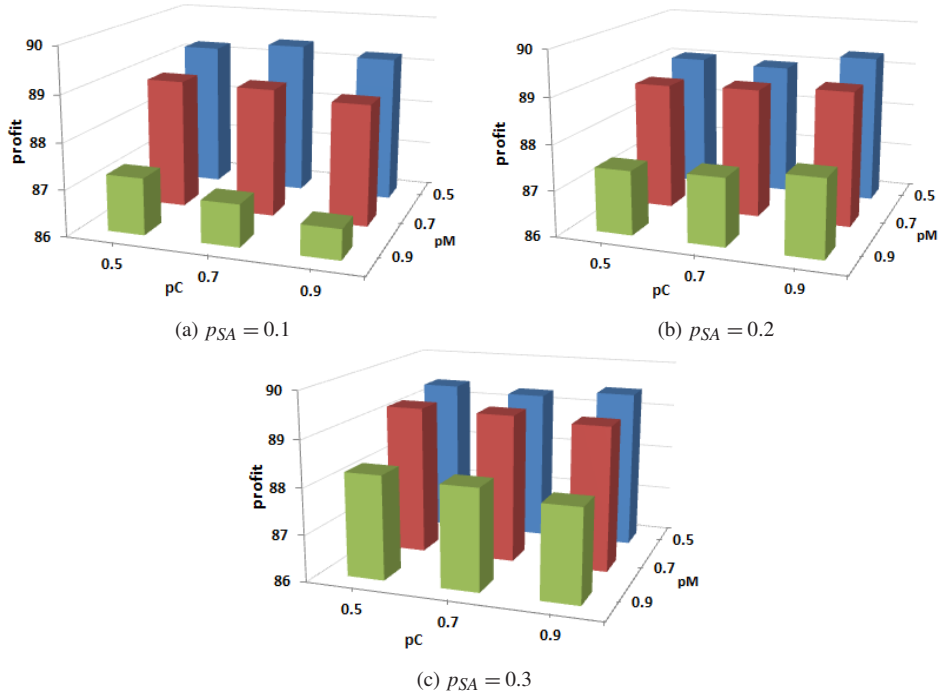
The graphics in Fig. 4 summarize the main results of the configuration analysis, reporting the average profit computed by the EA+SA algorithm when using each of the evaluated parameter settings for p_C, p_M, and p_{SA}.

6.4. *Results and Discussion*

This subsection reports and analyzes the main results of the experimental evaluation of the proposed metaheuristic methods to solve VMMP. The results computed using metaheuristics are compared against two different profit-greedy list-scheduling heuristics for the VMMP: Shortest Resource Cheapest Instance (SRCI) and Cheapest Instance (CI), already introduced in our previous work (Nesmachnow *et al.*, 2013).

6.4.1. *Methodology and Metrics*

A number of fifty independent executions were performed on each VMMP instance for both the sequential and parallel versions of each studied metaheuristic. The experimental analysis takes into account the VMMP objective of maximizing the broker profit, but



(a) $p_{SA} = 0.1$ (b) $p_{SA} = 0.2$

(c) $p_{SA} = 0.3$

Fig. 4. Summary of profit results for p_c, p_m, and p_{SA} parameter setting analysis.

we are also interested in standard metrics for planning and scheduling problems (i.e., *makespan* and *flowtime*), since obtaining short and balanced schedules implies a better utilization of the available resources from the broker.

We also evaluate the computational efficiency of the parallel versions of the proposed metaheuristics and the benefit of using parallelism, by performing executions using different number of distributed subpopulations.

In order to further analyze the planning computed by the metaheuristics, we also evaluate the *makespan* and the *flowtime* metrics for each planning. The makespan is a system-related metric used to evaluate the resource utilization; it is defined as the time spent from the moment when the first VM request begins its execution until the last VM request finishes its execution. Lower values of the makespan metric means that the RIs will be able to attend more requests in a given period of time. The flowtime evaluates the sum of the finishing times for all VM requests. It is an important metric from the user point-of-view, since it reflects the response time of the system for a set of submitted requests (Leung *et al.*, 2004).

For each of the proposed algorithms, we define the GAP metric as the relative difference on the makespan/flowtime required by each metaheuristic planning when compared to those computed by the CI or SCRI heuristic, according to Eq. (4) (GAP on makespan) and Eq. (5) (GAP on flowtime). In both equations MH is one of the three metaheuristic methods studied and H is the best heuristic (CI or SRCI) in terms of the corresponding (makespan/flowtime) metric:

$$\left(GAP_H^M = \frac{makespan_{MH} - makespan_H}{makespan_H}\right), \tag{4}$$

$$\left(GAP_H^F = \frac{flowtime_{MH} - flowtime_H}{flowtime_H}\right). \tag{5}$$

Finally, we also analyze the average relative number of *additional on-demand VMs* required by each metaheuristic. These additional VMs correspond to those requests that cannot be executed on the specified deadline with the available resources, thus an on-demand instance is required to be bought by the broker to execute the request.

6.4.2. *Numerical Results and Analysis*

Table 2 reports the best and average profit found by the studied algorithms, and the average profit improvement over the results computed by the best known list-scheduling heuristic for the VMMP. In fact, row "*over* (Nesmachnow *et al.*, 2013)" in Table 2 indicates how many times the corresponding algorithm outperformed the best list heuristic from our previous work (Nesmachnow *et al.*, 2013), regarding the profit value. Row #*best* indicates how many times the corresponding algorithm outperformed all the other studied algorithms in this paper, regarding the profit value. Table 2 reports results for the sequential

Table 2
Profit results for the proposed metaheuristics.

n	Metric	Metaheuristic								
		Sequential (1 deme)			Parallel: 8 demes			Parallel: 24 demes		
		EA+SA	GA	SA	EA+SA	GA	SA	EA+SA	GA	SA
50	Profit (avg.)	43.76	43.52	42.65	**43.78**	43.63	43.05	**43.78**	43.68	43.08
	Profit (best)	**56.75**	**56.75**	**56.75**	**56.75**	**56.75**	**56.75**	**56.75**	**56.75**	**56.75**
	Improv. (avg.)	**23.19%**	22.33%	19.05%	22.28%	22.79%	20.39%	22.29%	22.95%	20.52%
	Over[a]	**90**/100	**90**/100	87/100	**90**/100	**90**/100	89/100	**90**/100	**90**/100	89/100
	# best	**90**	56	19	**90**	59	48	**90**	60	53
100	Profit (avg.)	72.14	71.19	68.37	72.49	71.83	70.23	**72.62**	72.07	70.59
	Profit (best)	**92.94**	**92.94**	91.73	**92.94**	**92.94**	92.91	**92.94**	**92.94**	**92.94**
	Improv. (avg.)	53.51%	50.71%	42.67%	54.81%	52.93%	47.36%	**55.32%**	53.76%	48.30%
	Over[a]	**100**/100	**100**/100	**100**/100	**100**/100	**100**/100	**100**/100	**100**/100	**100**/100	**100**/100
	# best	**100**	6	0	**100**	26	7	**100**	27	15
200	Profit (avg.)	77.52	73.53	74.37	78.90	76.43	76.81	**79.21**	77.05	77.48
	Profit (best)	129.17	125.67	126.18	**130.03**	128.93	128.25	**130.03**	129.99	128.82
	Improv. (avg.)	94.64%	72.24%	84.51%	105.86%	83.54%	93.41%	**109.08%**	87.09%	95.81%
	Over[a]	93/100	90/100	93/100	**95**/100	92/100	93/100	**95**/100	92/100	93/100
	# best	89	1	7	**92**	0	5	**92**	0	5
400	Profit (avg.)	51.65	31.49	55.73	56.14	43.44	61.21	57.86	46.76	**62.79**
	Profit (best)	231.56	191.26	230.60	235.29	219.22	232.60	**237.22**	216.08	234.03
	Improv. (avg.)	328.58%	220.14%	363.36%	355.35%	291.15%	397.88%	368.88%	311.00%	**412.64%**
	Over[a]	78/100	73/100	80/100	79/100	75/100	78/100	**81**/100	75/100	76/100
	# best	43	0	44	**45**	0	42	42	1	44
All	Profit (avg.)	61.27	54.93	60.28	62.83	58.83	62.83	63.37	59.89	**63.48**
	Profit (best)	231.56	191.26	230.60	235.29	219.22	232.60	**237.22**	216.08	234.03
	Improv. (avg.)	124.98%	91.35%	127.39%	134.82%	112.60%	139.76%	139.14%	118.70%	**144.32%**
	Over[a]	361/400	353/400	360/400	364/400	357/400	360/400	**366**/400	357/400	358/400
	# best	322	63	70	**327**	85	102	324	88	117

[a] (Nesmachnow *et al.*, 2013).

Table 3
Makespan and flowtime results for the proposed parallel metaheuristics using 24 demes.

n	Metric	Metaheuristic					
		GAP_H^M			GAP_H^F		
		EA+SA	GA	S	EA+SA	GA	SA
50	Average	25.98%	21.69%	27.25%	44.41%	37.82%	51.10%
	Std. dev.	21.30%	17.84%	24.31%	9.72%	6.70%	12.48%
100	Average	20.87%	19.02%	21.39%	46.80%	43.58%	50.42%
	Std. dev.	11.59%	10.85%	13.84%	10.66%	9.68%	14.50%
200	Average	10.28%	9.56%	10.88%	49.15%	46.27%	48.57%
	Std. dev.	6.35%	6.71%	7.22%	21.41%	19.62%	22.54%
400	Average	10.76%	8.24%	11.88%	25.99%	24.60%	28.64%
	Std. dev.	6.08%	5.86%	5.98%	27.70%	22.27%	25.36%

algorithms and two different parallel versions, using 8 and 24 processing cores, grouped by the number of VM requests in the VMMP workloads (n). All averages are computed considering the fifty independent executions performed for each metaheuristic on each VMMP instance studied. The best results for each problem instance are emphasized in **bold font**.

The numerical results in Table 2 demonstrate that the parallel versions of the proposed metaheuristics clearly outperform the sequential ones, especially when using 24 demes (overall) and 8 demes to solve the large dimension problem instances tackled. Regarding the algorithms comparison, the parallel hybrid EA+SA was the best method among the studied ones for all problem instances, with the exceptions of a few workloads with 400 VM requests (where the parallel SA achieved similar profit results and better average improvements in some cases).

The parallel hybrid EA+SA outperformed the previous results computed with list scheduling heuristics on (Nesmachnow *et al.*, 2013) on 366 out of 400 problem instances studied. Average improvements of up to 368.88% were obtained in the profit values, for the problem instances with 400 requests.

The last five rows in Table 2 provide the average results of the algorithms on all problem instances studied in the experimental analysis. EA+SA found the best overall profit values for all instances of the four different VM request sizes. It outperforms the existing list heuristics in 91.5% of the 400 studied instances, and it is the metaheuristic outperforming the other studied ones in most cases (higher than three times more, in average, than the second best algorithm), for the sequential and the two different parallel versions. The experimental results demonstrate how useful is to combine the evolutionary search and the SA operator in a weak hybrid method, which is able to outperform both algorithms executing alone.

After evaluating the proposed algorithms in terms of profit, we are interested also on analyzing the Quality of Service (QoS) of the solutions they provide. Therefore, we study the average makespan and flowtime results for the parallel algorithms using 24 demes in Table 3. Results are computed considering the GAP metrics as defined in Eqs. (4) and (5).

Table 4
Load characterization of the VMMP instances regarding the number of requests over number of machines ratio.

# Machines	# Requests			
	50	100	200	400
10	Medium	Medium	High	High
20	Low	Medium	Medium	High
30	Low	Low	Medium	High
50	Low	Low	Low	Medium

We would like to remark at this point that the makespan and flowtime metrics are in conflict with the profit. The reason is that in order to improve profit we must assign every VM to the smallest (i.e., the cheapest) available RI that meets its requests, even when it implies a delay on the VM deployment due to a temporal unavailability of the desired RI. Such delays will penalize the makespan and flowtime measurements. Taking into account not only the profit but the QoS of the solution at the same time might be very interesting for the broker. This would imply the use of multi-objective optimization techniques. This is an interesting line of research that we highlight in the future works section.

The numerical results in Table 3 indicate that the algorithm providing the best profit, namely EA+SA, is also providing highly accurate QoS values, according to both makespan and flowtime. GA is the algorithm finding the solutions with best QoS in all cases (but it is the worst one according to profit metric). The analysis demonstrates that EA+SA provides accurate QoS values, close to those found by GA. The difference between these algorithms range from 5.85% (for 200 VM requests and flowtime) to 23.42% for the 400 VM requests and makespan.

In order to provide a deeper analysis of different planning situations modeled by the VMMP instances tackled, we characterized the instances according to the load ratio of each workload-scenario pair. The load ratio represents the stress intensity of the cloud infrastructure owned by the broker (i.e., how much stressed it is) in each problem scenario, and it is defined as $R_{load} = {}^{\#requests}/_{\#machines}$. In our analysis, we defined three load ratio classes: a problem instance is considered to have *low* load when $R_{load} \in [0, 5)$; it has a *medium* load when $R_{load} \in [5, 10]$; and it has a *high* load when $R_{load} \in (10, +\infty)$. Table 4 shows all the considered problem instances characterized according to this criterion.

Although a *medium* load is to be expected in common situations during most of the operational time for the cloud system, *low* and *high* loads are considered to occur in short bursts during bounded periods of time, depending on the users activity. The previously commented characterization is useful to analyze how much profitable the proposed subletting model is during both normal and extreme conditions.

Table 5 reports the average improvement on the broker profit (against the best previously existing results; Nesmachnow *et al.*, 2013) and the average number of additional on-demand VMs required ($\#_{OD}$) for the plannings computed by the three parallel metaheuristics when using 24 demes. The variation of on-demand VMs (Δ_{OD}) is introduced in order to compare the number of additional on-demand VMs required by the proposed metaheuristics with respect to the solutions found by the CI and SCRI heuristics. For each of the parallel metaheuristics studied, the Δ_{OD} metric is defined as the difference between

Table 5
Average number of additional on-demand VMs required and profit improvement for the proposed metaheuristics using 24 demes.

Load	Parallel metaheuristic (24 demes)								
	EA+SA			GA			SA		
	Profit improv.	Δ_{OD}	$\#_{OD}$	Profit improv.	Δ_{OD}	$\#_{OD}$	Profit improv.	Δ_{OD}	$\#_{OD}$
Low	41.67%	−1.02	0.06	29.37%	−0.98	0.10	34.35%	−1.01	0.07
Medium	58.63%	−8.06	12.60	54.63%	−3.78	16.89	56.03%	−5.61	15.06
High	412.28%	61.69	199.51	354.19%	60.27	198.09	460.29%	58.39	196.21

Table 6
Summary of the parallel EA+SA results using 24 demes: average and best profit, profit improvement and makespan gap over the best heuristic, and average ratio of additional on-demand VMs required.

n	Profit			Profit improv.	Makespan		Additional on-demand VMs
	Best	Avg.	# Best	Avg.	GAP^{M}_{CI}	GAP^{M}_{SRCI}	Avg. ratio
50	56.75	43.76	$^{90}/_{100}$	22.29%	4.11%	24.81%	0.66%
100	92.94	72.62	$^{100}/_{100}$	55.32%	10.61%	15.54%	7.35%
200	130.03	79.21	$^{95}/_{100}$	109.08%	6.35%	7.47%	14.68%
400	237.22	57.86	$^{81}/_{100}$	368.88%	8.79%	7.03%	45.37%

the number of additional on-demand VMs required by that metaheuristic and the heuristic which used the lowest number of them in its solution.

The results on Table 5 show how the average number of on-demand VMs required quickly increases with the VM requests load for the three studied algorithms. Indeed, for the highest load instances, it can be seen that the metaheuristics find solutions with higher number of on-demand VMs than the previously existing methods. However, the profit improvement values of the solutions provided by the metaheuristics are much higher that those provided by the algorithms in Nesmachnow *et al.* (2013), raising up to 460.29%. As it happened with the number of additional on-demand VMs, there is a quick increase on the profit improvement when the load increases. For both the low and medium load instances, the three metaheuristics find solutions with lower number of additional on-demand VMs than the heuristics.

We would like to recall that using additional on-demand VMs implies a decrease on the broker profit, because it will have to pay for these on-demand VMs to deploy the corresponding virtual machine on time. Therefore, this will positively impact on the QoS offered, because the VM will be deployed immediately, without any delay.

Previous results show that the parallel hybrid EA+SA algorithm is the best of the studied methods, outperforming the remaining ones in most of the studied problem instances. Table 6 presents a summary of the results computed by the EA+SA algorithm. In average, it improves the profit computed by the best heuristic in all of the evaluated instances: with average values ranging from 22.29% up to 368.88%. The average profit improvement steadily increases with the problem dimension, computing the best profit improvements when tackling the higher dimension instances. When comparing with the other metaheuristics, the EA+SA algorithm is able to compute the best profit results for

Table 7
Average profit improvement of EA+SA over the best heuristic, for 1, 8, and 24 demes.

n	Average profit improvement		
	1 deme	8 demes	24 demes
50	23.19%	23.28%	23.29%
100	53.51%	54.81%	55.32%
200	94.64%	105.86%	109.08%
400	328.58%	355.35%	368.88%

366 of the 400 problem instances; computing the best profit for at least 80 of the 100 problem instances for each dimension.

In terms of makespan, Table 6 shows that EA+SA provides slightly worse results than the heuristics (between 4.11% and 24.81%). The reason is that a more efficient use of the available RIs resources increases the number of requests assigned to each resource, which impacts negatively in the makespan of the schedule.

We also investigated the benefits of the parallel model in the EA+SA algorithm in order to compute more accurate solutions when additional computing resources are available. Table 7 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.

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, obtaining the best improvement when tackling the largest problem instances. The improvement for the 24 demes algorithm with respect to the one using 1 deme ranges from 0.1% for the smallest instances to 40.3% for the largest ones.

7. Conclusions and Future Work

This article presents three new parallel metaheuristics for the problem of virtual machines mapping in the cloud. The problem arises for the cloud broker that sublets reserved instances on a number of clouds 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 proposed in Nesmachnow *et al.* (2013) and Iturriaga *et al.* (2013). This paper is an extension of those works.

The new proposed algorithms are shown to clearly outperform the best existing results in the literature (Nesmachnow *et al.*, 2013) in an affordable amount of time. The profit of the broker is increased by up to 44.32% in average for all the studied instances when using the proposed techniques, which only require 90 seconds execution time. Additional scalability tests showed that the profit improves when increasing the computational effort (by using more computing elements in parallel), on all problem instances.

The solutions provided by the algorithms were further analyzed in terms of their Quality of Service (QoS). It was not considered in the optimization process, and indeed it is in conflict with the profit optimization, but it is certainly a feature to take into consideration in our solutions. The results pointed out EA+SA, the proposed hybrid method, as the best algorithm in terms of profit, while providing solutions of high QoS, comparable to the best ones. The proposed metaheuristics clearly outperformed the best previous existing results in terms of profit (up to 412% improvement), at the cost of slightly worse QoS values (10% in average).

The main lines for future work include to further analyze the behavior and dynamics of the new techniques (e.g., convergence and diversity properties), as well as to investigate other more accurate methods. Additionally, designing and solving a multi-objective version of the problem, accounting for profit and QoS maximization will be very interesting for the broker. Finally, designing a reliable forecasting technique to predict the resources the broker will need in the future is another important line of future research.

Acknowledgements. The work of S. Iturriaga and S. Nesmachnow was partially funded by ANII and PEDECIBA, Uruguay. B. Dorronsoro acknowledges the support offered by the National Research Fund, Luxembourg, AFR contract No. 4017742. E.-G. Talbi and P. Bouvry acknowledge that this study is partially supported by the CNRS, France, and the National Research Fund, Luxembourg, through Green@Cloud project, INTER/CNRS/11/03. This study was partially supported by the FNR (Luxembourg) and NCBiR (Poland), through IShOP project, INTER/POLLUX/13/6466384.

References

Alba, E., Luque, G., Garcia-Nieto, J., Ordoñez, G. (2007). MALLBA: a software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications*, 1(1), 74–85.

Alba, E., Luque, G., Nesmachnow, S. (2013). Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20(1), 1–48.

Bäck, T., Fogel, D., Michalewicz, Z. (Eds.) (1997). *Handbook of Evolutionary Computation*. Oxford University Press, Oxford.

Besis, N., Sotiriadis, S., Xhafa, F., Asimakopoulou, E. (2014). Cloud scheduling optimization: a reactive model to enable dynamic deployment of virtual machines instantiations. *Informatica*, 24(3), 357–380.

Blum, C., Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268–308.

Buyya, R., Broberg, J., Goscinski, A. (2011). *Cloud Computing: Principles and Paradigms*. Wiley, New York.

Buyya, R., Ranjan, R., Calheiros, R. (2010). Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. In: *10th International Conference on Algorithms and Architectures for Parallel Processing*, Busan, Korea, pp. 13–31.

Calheiros, R., Buyya, R. (2012). Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds. In: *13th International Conference on Web Information System Engineering*, Paphos, Cyprus, pp. 28–30.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1), 41–51.

Foster, I., Zhao, Y., Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In: *Grid Computing Environments Workshop*, pp. 1–10.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.

Gropp, W. (2002). MPICH2: a new start for MPI implementations. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, pp. 37–42.

Grozev, N., Buyya, R. (2012). Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, pp. 1–22, online first December 2012.

Iturriaga, S., Nesmachnow, S., Dorronsoro, B., Talbi, E.-G., Bouvry, P. (2013). A parallel hybrid evolutionary algorithm for the optimization of broker virtual machines sublet in cloud systems. In: *II Workshop on Soft Computing Tecnhiques in Cluster and Grid Computing Systems (SCCG 2013)*. Compiegne, France, pp. 594–599.

Kellerer, H., Pferschy, U., Pisinger, D. (2004). *Knapsack Problems*. Springer, Berlin.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.

Lampe, U. (2011). Optimizing the distribution of software services in infrastructure clouds. In: *IEEE World Congress on Services*, pp. 69–72.

Legillon, F., Melab, N., Renard, D., Talbi, E.-G. (2013). Cost minimization of service deployment in a multi-cloud environment. In: *IEEE International Conference on Evolutionary Computation*. Cancún, Mexico, pp. 2580–2587.

Leung, J., Kelly, L., Anderson, J. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton.

Mattess, M., Vecchiola, C., Garg, S.K., Buyya, R. (2011). *Cloud Computing: Methodology, Systems, and Applications*, Chapter: *Cloud Bursting: Managing Peak Loads by Leasing Public Cloud Services*. CRC Press, Boca Raton, pp. 343–368.

Nesmachnow, S. (2014). *Metaheuristics as soft computing techniques for efficient optimization*. In: Khosrow-Pour M. (Ed.), *Encyclopedia of Information Science and Technology*. IGI Global Publisher, Hershey.

Nesmachnow, S., Cancela, H., Alba, E. (2010). Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, 15(4), 685–701.

Nesmachnow, S., Cancela, H., Alba, E. (2012). A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing*, 12(2), 626–639.

Nesmachnow, S., Iturriaga, S., Dorronsoro, B., Talbi, E.-G., Bouvry, P. (2013). List scheduling heuristics for virtual machine mapping in cloud systems. In: *VI High Performance Computing Latin America Symposium*, Mendoza, Argentina, pp. 37–48.

Rimal, B., Choi, E., Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In: *5th International Joint Conference on INC, IMS and IDC (NCM)*, Seoul, Korea, pp. 44–51.

Rogers, O., Cliff, D. (2012). A financial brokerage model for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(2), 1–12.

Sotiriadis, S., Bessis, N., Antonopoulos, N. (2011). Towards inter-cloud schedulers: a survey of meta-scheduling approaches. In: *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. Barcelona, Spain, pp. 59–66.

Sotiriadis, S., Bessis, N., Antonopoulos, N. (2012). Decentralized meta-brokers for inter-cloud: Modeling brokering coordinators for interoperable resource management. In: *9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. Chongqing, Sichuan, China, pp. 2462–2468.

Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5), 541–564.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley, New York.

Tang, C., Steinder, M., Spreitzer, M., Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. In: *16th international conference on World Wide Web*. Banff, Canada, pp. 331–340.

Tordsson, J., Montero, R., Moreno-Vozmediano, R., Llorente, I. (2012). Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2).

Wu, F., Zhang, L., Huberman, B. (2008). Truth-telling reservations. *Algorithmica*, 52(1), 65–79.

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., Abraham, A. (2008). *Efficient batch job scheduling in grids using cellular memetic algorithms*. *Studies in Computational Intelligence*, Vol. 146. Springer, Berlin, pp. 273–299.

S. Nesmachnow is a Full Time Professor at Universidad de la República, Uruguay, and researcher in parallel metaheuristics and scientific high-performance computing in ANII and PEDECIBA, Uruguay. He has a PhD in Computer Science (2010) from Universidad de la República, Uruguay. He has published many papers in impact journals and proceedings and served in technical committees of international conferences. He is Editor-in-Chief of *International Journal of Metaheuristics*, Inderscience Publishers.

S. Iturriaga is an Assistant at Engineering Faculty, Universidad de la República, Uruguay, where he got his MSc degree in Computer Science (2013). He works on high-performance computing and parallel metaheuristics for scheduling. He published in four journals and many conferences in these areas.

B. Dorronsoro has a PhD in Computer Science (2007) from the University of Málaga, and he is currently working at University of Lille. His main research interests include Grid and Cloud computing, MANETs, and optimization using meta-heuristics. He has several articles in impact journals and two books.

E.-G. Talbi received the Master and PhD degrees in Computer Science from the Institut National Polytechnique de Grenoble in France. He is a full Professor at the University of Lille and the head of DOLPHIN research group from both the Lille's Computer Science laboratory (LIFL, Université Lille 1) and INRIA Lille Nord Europe. His current research interests are in the field of multi-objective optimization, parallel algorithms, metaheuristics, combinatorial optimization, cluster and grid computing, hybrid and cooperative optimization, and applications to logistics/transportation, bioinformatics and networking. Professor Talbi has to his credit more than 100 international publications including journal papers, book chapters and conference proceedings.

P. Bouvry has a PhD (1994) in Computer Science at the University of Grenoble, France. He is a Professor at the University of Luxembourg. Pascal Bouvry is specialized in parallel and evolutionary computing. His current interest concerns the application of nature-inspired computing for solving reliability, security, and energy-efficiency problems

Metaeuristikos virtualiųjų mašinų planavimo debesyse uždaviniui

Sergio NESMACHNOW, Santiago ITURRIAGA, Bernabé DORRONSORO, El-Ghazali TALBI, Pascal BOUVRY

Šiame straipsnyje pristatomos nuosekliosios ir lygiagrečiosios metaeuristikos virtualiųjų mašinų subnuomos debesų sistemose uždaviniui spręsti, kuris susijęs su virtualiųjų mašinų prašymų paskyrimu iš debesų tarpininko užsakytiems resursams maksimizuojant tarpininko pelną. Trys metaeuristikos yra tiriamos: atkaitinimo modeliavimas, genetinis algoritmas ir hibridinis evoliucinis algoritmas. Eksperimentinis įvertinimas naudojant realius darbų srautų ir scenarijų pavyzdžių duomenis iš debesų paslaugų teikėjų rodo, kad lygiagretusis hibridinis evoliucinis algoritmas yra labiausiai tinkamas šiam uždaviniui spręsti.