Towards Elastic Virtual Machine Placement in Overbooked OpenStack Clouds under Uncertainty

Fabio López-Pires[1], Benjamín Barán[2], Carolina Pereira[2], Marcelo Velázquez[2], and Osvaldo González[2]

[1]*Itaipu Technological Park, Hernandarias, Paraguay*
fabio.lopez@pti.org.py
[2]*National University of the East, Ciudad del Este, Paraguay*
{bbaran,cpereira,mvelazquez,ogonzalez}@fpune.edu.py

Abstract

Cloud computing datacenters currently provide millions of virtual machines in highly dynamic Infrastructure as a Service (IaaS) markets. As a first step on implementing algorithms previously proposed by the authors for Virtual Machine Placement (VMP) in a real-world IaaS middleware, this work presents an experimental comparison of these algorithms against current algorithms considered for solving VMP problems in OpenStack. Several experiments considering scenario-based simulations for uncertainty modelling demonstrate that the proposed algorithms present promising results for its implementation towards real-world operations. Next research steps are also summarized.

Keywords: Virtual Machine Placement, OpenStack, Multi-Objective Optimization, Cloud Datacenters.

1 Introduction

This work focuses on a well-known problem: the process of selecting which requested virtual machines (VMs) should be hosted at each available physical machine (PM) of a cloud computing infrastructure, denoted in the specialized literature as Virtual Machine Placement (VMP). A previously proposed complex Infrastructure as a Service (IaaS) environment for VMP problems is considered, taking into account service elasticity and overbooking of physical resources [?].

In this context, this work also considers a previously proposed two-phase optimization scheme, decomposing the VMP problem into two different sub-problems, combining advantages of online (incremental VMP or iVMP) and offline (VMP reconfiguration or VMPr) VMP formulations. This is mainly because online decisions made along the operation of a dynamic cloud computing infrastructure negatively affects the quality of obtained solutions in VMP problems when comparing to offline decisions [?]. Unfortunately, offline VMP formulations are not appropriate for highly dynamic real-world IaaS environments, where cloud services are requested according to current demand.

When studying a two-phase optimization scheme for VMP problems, additional considerations should be analysed, e.g. methods to decide when or under what circumstances to trigger placement reconfigurations with migration of VMs between PMs (*VMPr Triggering*) and what to do with cloud services requested during placement recalculation (*VMPr Recovering*).

Due to the randomness of customer requests, VMP problems should be formulated under uncertainty [?]. This work considers a scenario-based uncertainty approach for modeling relevant uncertain parameters.

Taking into account experimental results already obtained in simulations against state-of-the-art alternative approaches for VMP problems considering 400 experimental scenarios, the implementation of the already proposed algorithms in a real-world IaaS middleware is a natural continuation of the work presented in [?]. As a previous step of the mentioned implementation, this work considers a previously developed Dynamic VMP Framework[1] for extending simulations including current VMP algorithms considered in OpenStack. The official OpenStack Filter Scheduler algorithm was slightly adapted to fit into the considered formulation, as described in the following sections.

The remainder of this paper is structured in the following way: Section **??** presents the considered uncertain VMP problem formulation, while Section **??** presents details on the design and implementation of evaluated alternatives to solve the formulation of the VMP problem. Section **??** summarize experimental results. Conclusions and future work are left to Section **??**.

2 Considered VMP Formulation

This section summarizes the considered VMP formulation under uncertainty previously proposed by some of the authors in [?]. This VMP formulation is based on a two-phase scheme for the optimization of the following objective functions: (i) power consumption,

[1]http://*github.com/DynamicVMP*

(ii) economical revenue, (iii) resource utilization and (iv) placement reconfiguration time.

According to the taxonomy presented in [?], this work focuses on a provider-oriented VMP for federated-cloud deployments, considering a combination of two types of formulations: (i) online (i.e. iVMP) and (ii) offline (i.e. VMPr). Interested readers may refer to [?] for more details on the motivation of using a two-phase optimization scheme as well as more details on the VMP formulation itself that are not included in this work due to space limitations.

The following sub-sections summarize the most relevant details on the considered uncertain VMP formulation previously proposed in [?].

2.1 Complex IaaS Environment

The considered formulation of the VMP problem models a complex IaaS environment, composed by available PMs and VMs requested at each discrete time t, considering the following information as input data for the proposed VMP problem:

- a set of n available PMs and specifications (**??**);
- a set of $m(t)$ VMs requested, at each discrete time t, and specifications (**??**);
- information about the utilization of resources of each active VM at each discrete time t (**??**);
- current placement at each discrete time t (i.e. $x(t)$) (**??**).

The iVMP and VMPr sub-problems consider different sub-sets of the above mentioned input data, as presented later in Sections **??** and **??**.

The set of PMs owned by the IaaS provider is represented as a matrix $H \in \mathbb{R}^{n\times(r+2)}$, as presented in (**??**). Each PM H_i is represented by r different physical resources. This work considers $r = 3$ physical resources (Pr_1 to Pr_3): CPU [EC2 Compute Unit (ECU)], RAM [GB] and network capacity [Mbps]. The maximum power consumption [W] is also considered. Finally, considering that an IaaS provider could own more than one cloud datacenter, PMs notation also includes a datacenter identifier c_i, i.e.

$$H = \begin{bmatrix} Pr_{1,1} & \dots & Pr_{r,1} & pmax_1 & c_1 \\ \dots & \dots & \dots & \dots & \dots \\ Pr_{1,n} & \dots & Pr_{r,n} & pmax_n & c_n \end{bmatrix} \tag{1}$$

where:

$Pr_{k,i}$:	Physical resource k on H_i, where $1 \leq k \leq r$;
$pmax_i$:	Maximum power consumption of H_i in [W];
c_i:	Datacenter identifier of H_i, where $1 \leq c_i \leq c_{max}$;
n:	Total number of PMs.

In this context, the IaaS provider dynamically receives requests of cloud services for placement (i.e. a set of inter-related VMs) at each discrete time t. A cloud service S_b is composed by a set of VMs.

The set of VMs requested by customers at each discrete time t is represented as a matrix $V(t) \in \mathbb{R}^{m(t)\times(r+2)}$, as presented in (**??**). In this work, each VM V_j requires $r = 3$ different virtual resources ($Vr_{1,j}(t)$-$Vr_{3,j}(t)$): CPU [ECU], RAM memory [GB] and network capacity [Mbps]. Additionally, a cloud service identifier b_j is considered, as well as an economical revenue R_j [\$] associated to each VM V_j.

$$V(t) = \begin{bmatrix} Vr_{1,1}(t) & \dots & Vr_{r,1}(t) & b_1 & R_1(t) \\ \dots & \dots & \dots & \dots & \dots \\ Vr_{1,m(t)}(t) & \dots & Vr_{r,m(t)}(t) & b_m(t) & R_{m(t)}(t) \end{bmatrix} \tag{2}$$

where:

$Vr_{k,j}(t)$:	Virtual resource k on V_j, where $1 \leq k \leq r$;
b_j:	Service identifier of V_j;
$R_j(t)$:	Economical revenue for allocating V_j in [\$] at instant t;
$m(t)$:	Number of VMs at each discrete time t, where $1 \leq m(t) \leq m_{max}$;
m_{max}:	Maximum number of VMs.

To model a dynamic VMP environment taking into account both vertical and horizontal elasticity of cloud services, as previously presented in [?], the set of requested VMs $V(t)$ may include the following types of requests for cloud service placement at each time t:

- **cloud services creation:** where new a cloud service S_b, composed by one or more VMs V_j, is created. Consequently, the number of VMs at each discrete time t (i.e. $m(t)$) is a function of time;
- **scale-up / scale-down of VMs resources:** where one or more VMs V_j of a cloud service S_b increases (scale-up) or decreases (scale-down) its capacities of virtual resources with respect to current demand (vertical elasticity). In order to model these considerations, virtual resource capacities of a VM V_j (i.e. $Vr_{1,j}(t)$-$Vr_{3,j}(t)$) are a function of time, as well as the associated economical revenue ($R_j(t)$);
- **cloud services scale-out / scale-in:** where a cloud service S_b increases (scale-out) or decreases (scale-in) the number of associated VMs according to current demand (horizontal elasticity). Consequently, the number of VMs V_j in a cloud service S_b at each discrete time t, denoted as $mS_b(t)$, is a function of time;
- **cloud services destruction:** where virtual resources of cloud services S_b, composed by one or more VMs V_j, are released.

Resource utilization of each VM V_j at each discrete time t is represented as a matrix $U(t) \in \mathbb{R}^{m(t)\times r}$, as presented in (**??**):

$$U(t) = \begin{bmatrix} Ur_{1,1}(t) & \dots & Ur_{r,1}(t) \\ \dots & \dots & \dots \\ Ur_{1,m(t)}(t) & \dots & Ur_{r,m(t)}(t) \end{bmatrix} \quad (3)$$

where:

$Ur_{k,j}(t)$: Utilization ratio of $Vr_k(t)$ in V_j at each discrete time t.

The current placement of VMs into PMs $(x(t))$ represents VMs requested in the previous discrete time $t-1$ and assigned to PMs; consequently, the dimension of $x(t)$ is based on the number of VMs $m(t-1)$. The placement at each discrete time t is represented as a matrix $x(t) \in \{0,1\}^{m(t-1)\times n}$, as defined in (**??**):

$$x(t) = \begin{bmatrix} x_{1,1}(t) & x_{1,2}(t) & \dots & x_{1,n}(t) \\ \dots & \dots & \dots & \dots \\ x_{m(t-1),1}(t) & x_{m(t-1),2}(t) & \dots & x_{m(t-1),n}(t) \end{bmatrix} \quad (4)$$

where:

$x_{j,i}(t) \in \{0,1\}$: indicates if V_j is allocated $(x_{j,i}(t) = 1)$ or not $(x_{j,i}(t) = 0)$ for execution in a PM H_i at time t (i.e. $x_{j,i}(t) : V_j \to H_i$).

2.2 Incremental VMP (iVMP)

In online algorithms for solving the considered VMP problem, placement decisions are performed at each discrete time t. The formulation of the considered iVMP (online) problem is based on [**?**] and could be formally enunciated as:

Given a complex IaaS environment composed by a set of PMs (H), a set of active VMs already requested before time t $(V(t))$, and the current placement of VMs into PMs (i.e. $x(t)$), it is sought an incremental placement of $V(t)$ into H for the discrete time $t+1$ $(x(t+1))$ *without migrations, satisfying the problem constraints and optimizing the considered objective functions.*

2.2.1 Input Data for iVMP

As presented in [**?**], the considered formulation of the iVMP problem receives the following information as input data:

- a set of n available PMs and specifications (**??**);
- a dynamic set of $m(t)$ requested VMs (already allocated VMs plus new requests) and specifications (**??**);
- information about the utilization of resources of each active VM at each discrete time t (**??**);
- current placement at each discrete time t (i.e. $x(t)$) (**??**).

2.2.2 Output Data for iVMP

The result of the iVMP phase at each discrete time t is an incremental placement $\Delta x(t)$ for the next time instant in such a way that $x(t+1) = x(t) + \Delta x(t)$. Clearly, the placement at $t+1$ is represented as a matrix $x(t+1) \in \{0,1\}^{m(t)\times n}$, as defined in (**??**):

$$x(t+1) = \begin{bmatrix} x_{1,1}(t+1) & x_{1,2}(t+1) & \dots & x_{1,n}(t+1) \\ \dots & \dots & \dots & \dots \\ x_{m(t),1}(t+1) & x_{m(t),2}(t+1) & \dots & x_{m(t),n}(t+1) \end{bmatrix} \quad (5)$$

Formally, the placement for the next time instant $x(t+1)$ is a function of the current placement $x(t)$ and the active VMs at discrete time t, i.e.:

$$x(t+1) = f\,[x(t), V(t)] \quad (6)$$

2.3 VMP Reconfiguration (VMPr)

As it was previously mentioned in [**?**] an offline algorithm solves a VMP problem considering a static environment where VM requests do not change over time and considers migration of VMs between PMs. The formulation of the proposed VMPr (offline) problem is based on [**?**, **?**] and could be enunciated as:

Given a current placement of VMs into PMs $(x(t))$, it is sought a placement reconfiguration through migration of VMs between PMs for the discrete time t (i.e. $x'(t)$), satisfying the constraints and optimizing the considered objective functions.

2.3.1 Input Data for VMPr

The proposed formulation of the VMPr problem receives the following information as input data:

- a set of n available PMs and specifications (**??**);
- information about the utilization of resources of each active VM at discrete time t (**??**);
- current placement at discrete time t (i.e. $x(t)$) (**??**).

2.3.2 Output Data for VMPr

The result of the VMPr problem is a placement reconfiguration through migration of VMs between PMs for the discrete time t (i.e. $x'(t)$), represented by:

- a placement reconfiguration of $x(t)$, i.e. $x'(t)$ (**??**);

Summarizing the considered constraints, a VM V_j must be allocated to run on a single PM H_i or alternatively located in another federated IaaS provider. It should be mentioned that from an IaaS provider perspective, elastic cloud services usually are considered more important than non-elastic ones. Consequently, resources of elastic cloud services most of the time are allocated with higher priority over non-elastic ones, what usually is reflected in the contracts between an IaaS provider and each customer. Additionally, a PM H_i must have sufficient available resources to meet the dynamic requirements of all VMs V_j that are allocated to run on H_i. It is important to remember that resources of VMs are dynamically used, giving space to re-utilization of idle resources that were already reserved. Re-utilization of idle resources could represent higher risk of unsatisfied demand in case utilization of resources increases in a short period of time. Therefore, providers need to reserve a percentage of idle resources as a protection (defined by a protection factor λ_k) in case overbooking is used.

2.4 Objective Functions

More than 60 different objective functions for VMP problems were already identified in [?, ?]. Considering the large number of existing objective functions, identified objective functions with similar characteristics and goals could be classified into 5 objective function groups [?]: (G1) energy consumption, (G2) network traffic, (G3) economical costs, (G4) resource utilization and (G5) performance.

As previously considered in [?], the optimization of four objective functions is taken into account. It is important to consider that by no means, the authors claim that the considered objective functions represent the best way to model VMP problems. This formulation only illustrates a reasonable formulation of a VMP problem in order to be able to study the main contributions of this work, considering the presented experimental evaluation of VMP algorithms.

In general, objective functions can be minimized while maximizing other objectives functions. In this work each considered objective function is formulated in a single optimization context (i.e. minimization).

2.4.1 Power Consumption Minimization

The power consumption minimization can be represented by the sum of the power consumption of each PM H_i that composes the complex IaaS environment (see Section **??**), as defined in (**??**).

$$f_1(x,t) = \sum_{i=1}^{n} ((pmax_i - pmin_i) \times Ur_{1,i}(t) + pmin_i) \times Y_i(t) \tag{7}$$

where:

x:	Evaluated solution of the problem;
$f_1(x,t)$:	Total power consumption of PMs at instant t;
$pmax_i$:	Maximum power consumption of a PM H_i;
$pmin_i$:	Minimum power consumption of a PM H_i; As suggested in [?], $pmin_i \approx pmax_i * 0.6$;
$Ur_{1,i}(t)$:	Utilization ratio of resource 1 (in this case CPU) by H_i at instant t;
$Y_i(t) \in \{0,1\}$:	Indicates if H_i is turned on $(Y_i(t) = 1)$ or not $(Y_i(t) = 0)$ at instant t.

2.4.2 Economical Revenue Maximization

Equation (**??**) represents leasing costs, defined as the sum of the total costs of leasing each VM V_j that is effectively allocated for execution on any PM of an alternative datacenter of the cloud federation. A provider must offer its idle resources to the cloud federation at lower prices than offered to customers in the actual cloud market for the federation to make sense. The pricing scheme may depend on the particular agreement between providers of the cloud federation [?]. For simplicity, this formulations considers that the main provider may lease requested resources (that are not able to provide) from the cloud federation at 70% $(\hat{X}_j = 0.7)$ of its market price $(R_j(t))$. These Leasing Costs $(LC(t))$ may be formulated as:

$$LC(t) = \sum_{j=1}^{m(t)} (R_j(t) \times X_j(t) \times \hat{X}_j) \tag{8}$$

where:

$LC(t)$:	Total leasing costs at instant t;
$R_j(t)$:	Economical revenue for attending V_j in [\$] at instant t;
$X_j(t) \in \{0,1\}$:	Indicates if V_j is allocated for execution on a PM $(X_j(t) = 1)$ or not $(X_j(t) = 0)$ at instant t;
$\hat{X}_j$:	Indicates if V_j is allocated on the main provider $(\hat{X}_j = 0)$ or on an alternative datacenter of the cloud federation $(\hat{X}_j = 0.7)$;
$m(t)$:	Number of VMs at each discrete time t, where $1 \le m(t) \le m_{max}$.

It is important to note that $\hat{X}_j$ is not necessarily a function of time. The decision of locating a VM V_j on a federated provider is considered only in the placement process, with no possible migrations between different IaaS providers.

Additionally, overbooked resources may incur in unsatisfied demand of resources at some periods of time, causing Quality of Service (QoS) degradation, and consequently Service Level Agreement (SLA) violations with economical penalties. These economical penalties should be minimized for an economical revenue maximization. Based on the workload independent QoS metric presented in [?], formalized in SLAs, Equation (**??**) represents total economical penalties for SLA violations, defined as the sum of the total penalties costs for unsatisfied demand of resources.

$$EP(t) = \sum_{j=1}^{m(t)} \left(\sum_{k=1}^{r} Rr_{k,j}(t) \times \Delta r_{k,j}(t) \times X_j(t) \times \phi_k \right) \tag{9}$$

where:

$EP(t)$:	Total economical penalties at instant t;
r:	Number of considered resources. In this paper 3: CPU, RAM memory and network capacity;

$Rr_{k,j}(t)$: Economical revenue for attending $Vr_{k,j}(t)$;
$\Delta r_{k,j}(t)$: Ratio of unsatisfied resource k at instant t where $\Delta r_{k,j}(t) = 1$ means no unsatisfied resource, while $\Delta r_{k,j}(t) = 0$ means resource k is unsatisfied in 100%;
$X_j(t) \in \{0, 1\}$: Indicates if V_j is allocated for execution on a PM ($X_j(t) = 1$) or not ($X_j(t) = 0$) at instant t;
ϕ_k: Penalty factor for resource k, where $\phi_k \geq 1$;
$m(t)$: Number of VMs at each discrete time t, where $1 \leq m(t) \leq m_{max}$.

In this work, the maximization of the total economical revenue that an IaaS provider receives is achieved by minimizing the total costs of leasing resources from alternative datacenters of the cloud federation as well as the total economical penalties for SLA violations, as presented in (**??**), i.e.

$$f_2(x,t) = LC(t) + EP(t) \tag{10}$$

where:

$f_2(x,t)$: Total economical expediture of the main IaaS provider at instant t.

2.4.3 Resources Utilization Maximization

This work considers a maximization of the resource utilization by minimizing the average ratio of wasted resources on each PM H_i (i.e. resources that are not allocated to any VM V_j).

$$f_3(x,t) = \frac{\sum_{i=1}^{n} \left[1 - \left(\frac{\sum_{k=1}^{r} Ur_{k,i}(t)}{r} \right) \right] \times Y_i(t)}{\sum_{i=1}^{n} Y_i(t)} \tag{11}$$

where:

$f_3(x,t)$: Average ratio of wasted resources at instant t;
$Ur_{k,i}(t)$: Utilization ratio of resource k of PM H_i at instant t;
r: Number of considered resources. In this paper $r = 3$: CPU, RAM memory and network capacity.

2.4.4 Reconfiguration Time Minimization

Inspired in [**?**], once a placement reconfiguration is accepted in the VMPr phase, all VM migrations are assumed to be performed in parallel through a management network exclusively used for these actions, increasing 10% CPU utilization in VMs being migrated. Consequently, the minimization of the (maximum) reconfiguration time could be achieved by minimizing the maximum amount of memory to be migrated from one PM H_i to another $H_{i'}$ $(i \neq i')$.

Equation (**??**) was proposed in [**?**] to minimize the maximum amount of RAM memory that must be moved between PMs at instant t.

$$f_4(x,t) = \max(MT_{i,i'}) \quad \forall i, i' \in \{1, \ldots, n\} \tag{12}$$

where:

$f_4(x,t)$: Network traffic overhead for VM migrations at instant t;
$MT_{i,i'}$: Total amount of RAM memory to be migrated from PM H_i to $H_{i'}$.

The following sub-section summarizes the main considerations taken into account to combine the four presented objective functions into a single objective function to be minimized with the aim of having a single figure of merit (or optimization metric).

2.5 Normalization and Scalarization

Each considered objective function must be formulated in a single optimization context (in this case, minimization) and each objective function cost must be normalized to be comparable and combinable as a single objective. This work normalizes each objective function cost by calculating $\hat{f}_i(x,t) \in \mathbb{R}$, where $0 \leq \hat{f}_i(x,t) \leq 1$ for each objective function $f_i(x,t)$.

$$\hat{f}_i(x,t) = \frac{f_i(x,t) - f_i(x,t)_{min}}{f_i(x,t)_{max} - f_i(x,t)_{min}} \tag{13}$$

where:

$\hat{f}_i(x,t)$: Normalized cost of objective function $f_i(x,t)$ at instant t;
$f_i(x,t)$: Cost of original objective function $f_i(x,t)$;
$f_i(x,t)_{min}$: Minimum possible cost for $f_i(x,t)$;
$f_i(x,t)_{max}$: Maximum possible cost for $f_i(x,t)$.

The presented normalized objective functions are combined into a single objective considering a minimum Euclidean distance to the origin, expressed as:

$$F(x,t) = \sqrt{\sum_{i=1}^{q} \hat{f}_i(x,t)^2} \tag{14}$$

where:

$F(x,t)$: Single objective function combining each $\hat{f}_i(x,t)$ at instant t;
$\hat{f}_i(x,t)$: Normalized cost of objective function $f_i(x,t)$ at instant t;
q: Number of objective functions.

2.6 Scenario-based Uncertainty Modeling

In this work, uncertainty is modeled through a finite set of well-defined scenarios S [**?**], where the following uncertain parameters are considered: (i) virtual resources capacities (vertical elasticity), (ii) number of VMs that compose cloud services (horizontal elasticity), (iii) utilization of CPU and RAM memory virtual resources and (iv) utilization of networking virtual resources (both relevant for overbooking).

For each scenario $s \in S$, a temporal average value of the objective function $F(x,t)$ presented in (**??**) is calculated as:

$$\overline{f_s(x,t)} = \frac{\sum_{t=1}^{t_{max}} F(x,t)}{t_{max}} \tag{15}$$

where:

$\overline{f_s(x,t)}$:	Temporal average of combined objective function for all discrete time instants t in scenario $s \in S$;
t_{max}:	Duration of a scenario in discrete time instants.

As previously described, when parameters are uncertain, it is important to find solutions that are acceptable for any (or most) considered scenario $s \in S$. This work considers minimization of the average objective function costs criteria [**?**] to select among solutions:

$$F_1 = \overline{F(x,t)} = \frac{\sum_{s=1}^{|S|} \overline{f_s(x,t)}}{|S|} \tag{16}$$

where:

F_1:	Average $\overline{f_s(x,t)}$ for all scenarios $s \in S$ [**?**].

3 Evaluated Algorithms

Considering a previous research work of some of the authors [**?**], promising results of the proposed algorithm were found in order to implement it in real-world IaaS middlewares. The mentioned proposed algorithm considers a two-phase optimization scheme using *First-Fit Decreasing* (FFD) for the iVMP phase, a *Memetic Algorithm* (MA) for the VMPr phase, a prediction-based method for VMPr Triggering and an update-based method for VMPr Recovering. This algorithm was denoted as *Algorithm 3 (A3)* in [**?**] and is considered in this work as *Algorithm 1 (A1)* for the presented experimental evaluation.

Additionally, and as a first step on implementing *A1* in a real-world IaaS middleware, official OpenStack algorithms for VMP were studied [**?**]. In this context, two alternatives are available for configuring VMP processes in OpenStack: (i) *Filter Scheduler* and (ii) *Random Scheduler*. Taking into account that the *Random Scheduler* uses a trivial logic for solving the VMP, this work considers the *Filter Scheduler* as *Algorithm 2 (A2)* for the presented experimental evaluation.
It is important to note that *A2* considers only the iVMP phase for its operation, without taking into account migration of VMs between PMs.

The following sub-sections briefly present some relevant aspects on evaluated algorithms *A1* and *A2*.

3.1 Algorithm 1: Two-Phase Optimization

This section presents details on algorithm *A1* [**?**] as considered iVMP and VMPr algorithms as well as considered VMPr Triggering and Recovering methods.

3.1.1 Incremental VMP (iVMP) for A1

In experimental results previously obtained by some of the authors in [**?**], the First-Fit Decreasing (FFD) heuristic outperformed other evaluated heuristics in average; consequently, the mentioned heuristic was considered in *A1* for the iVMP phase (see Table **??**). In the First-Fit (FF) heuristic, requested VMs $V_j(t)$ are allocated on the first PM H_i with available resources. The considered FFD heuristic operates similarly to FF heuristic, with the main difference that FFD heuristic sorts the list of requested VMs $V_j(t)$ in decreasing order by revenue $R_j(t)$ (see details in Algorithm **??**).

Taking into account the particularities of the proposed complex IaaS environment, the FFD heuristic presents some modifications when comparing to the one presented in [**?**], mainly considering the cloud service request types previously described in Section **??**. In fact, Algorithm **??** shows that cloud service destruction, scale-down of VM resources and cloud services scale-in are processed first, in order to release resources for immediate re-utilization (steps 1-3 of Algorithm **??**). At step 4, requests from $V(t)$ are sorted by a given criterion as revenue ($R_j(t)$) in decreasing order (of course, other criterion may be considered, as CPU [**?**]), where scale-up of VM resources and cloud services scale-out are firstly processed (steps 5-6), in order to consider elastic cloud services more important than non-elastic ones. Next, unprocessed requests from $V_j(t)$ include only cloud service creations that are allocated in decreasing order (steps 7-18). Here, a V_j is allocated in the first H_i with available resources after considering previously sorted $V(t)$. If no H_i has sufficient resources to host V_j, it is allocated in another federated provider. Finally, the placement $x(t+1)$ is updated and returned (steps 19-20).

3.1.2 VMP Reconfiguration (VMPr) for A1

Previous research work by the authors focused on developing VMPr algorithms considering centralized decisions such as the offline MAs presented in [**?**, **?**, **?**]. In this work, the considered VMPr algorithm for *A1* is based on the one presented in [**?**] and it works in the following way (see details in Algorithm **??**):

At step 1, a set Pop_0 of candidate solutions is randomly generated. These candidate solutions are repaired at step 2 to ensure that Pop_0 contains only feasible solutions, satisfying defined constraints.

Then, the algorithm tries to improve candidate solutions at step 3 using local search. With the obtained solutions, elitism is applied and the first best solution $x'(t)$ is selected from $Pop_0'' \cup x(t)$ at step 4 using objective function defined in (**??**). After an initialization in step 5, evolution begins (steps 6-12). The evolutionary process basically follows a similar behavior: solutions are selected from the union of the evolutionary set of solutions (or population), also known as Pop_u, and the best known solution $x'(t)$ (step 7), crossover and mutation operators are applied as usual (step 8), and

Algorithm 1: First-Fit Decreasing (FFD) for iVMP phase in Algorithm $A1$.

```
Data: $H$, $V(t)$, $U(t)$, $x(t)$ (see notation in Section ??)
Result: Incremental Placement $x(t+1)$
process cloud services destruction from $V(t)$;
process scale-down of VMs resources from $V(t)$;
process cloud services scale-in from $V(t)$;
sort VMs by revenue ($R_j(t)$) in decreasing order;
process scale-up of VMs resources from $V(t)$;
process cloud services scale-out from $V(t)$;
foreach unprocessed $V_j$ in $V(t)$ do
    while $V_j$ is not allocated do
        foreach $H_i$ in $H$ do
            if $H_i$ has enough resources to host $V_j$ then
                allocate $V_j$ into $H_i$ and break loop;
            end if
        end foreach
        if $V_j$ is still not allocated then
            allocate $V_j$ in another federated provider;
        end if
    end while
end foreach
update $x(t+1)$ with processed requests;
return $x(t+1)$
```

Algorithm 2: Memetic Algorithm (MA) for VMPr phase in Algorithm $A1$.

```
Data: $H$, $U(t)$, $x(t)$ (see notation in Section ??)
Result: Recalculated Placement $x'(t)$
initialize set of candidate solutions $Pop_0$;
$Pop'_0$ = repair infeasible solutions of $Pop_0$;
$Pop''_0$ = apply local search to solutions of $Pop'_0$;
$x'(t)$ = select best solution from $Pop''_0 \cup x(t)$
  considering (??);
$u = 0; Pop_u = Pop''_0$;
while stopping criterion is not satisfied do
    $Pop_u$ = selection of solutions from $Pop_u \cup x'(t)$;
    $Pop'_u$ = crossover and mutation on solutions of
      $Pop_u$;
    $Pop''_u$ = repair infeasible solutions of $Pop'_u$;
    $Pop'''_u$ = apply local search to solutions of $Pop''_u$;
    $x'(t)$ = select best solution from $Pop'''_u$ considering
      (??);
    increment number of generations $u$;
end while
return $x'(t)$
```

eventually solutions are repaired, as there may be infeasible solutions (step 9). Improvements of solutions of the evolutionary population Pop_u may be generated at step 10 using local search (local optimization). At step 11, the best known solution $x'(t)$ is updated (if applicable), while at step 12 the generation (or iteration) counter is updated. The evolutionary process is repeated until the algorithm meets a stopping criterion, returning the best known solution $x'(t)$ for a placement reconfiguration. More details may be found in [?].

Algorithm 3: Update-based VMPr Recovering in Algorithm $A1$.

```
Data: $x(t)$, $x'(t-\beta)$ (see notation in Section ??)
Result: Recovered Placement $x'(t)$
remove VMs $V_j$ from $x'(t-\beta)$ that are no longer
  running in $x(t)$
adjust resources from $x'(t-\beta)$ that changed in $x(t)$
add VMs $V_j$ from $x(t)$ that were not considered in
  $x'(t-\beta)$
if $x'(t-\beta)$ is better than $x(t)$ then ;
    return $x'(t-\beta)$;
else return $x(t)$ ;
```

3.1.3 Prediction-based Triggering for A1

In this work, *A1* considers a prediction-based method that analyses objective function (see (**??**)), in a way that it is possible to detect situations where a placement might be required for reconfiguration purposes.

The presented prediction-based VMPr Triggering method considers *Double Exponential Smoothing* (DES) [?] as a statistical technique for predicting values of the objective function $F(x,t)$, as formulated next in (**??**) to (**??**):

$$S_t = \alpha \times Z_t + (1-\tau)(S_{t-1} + b_{t-1}) \qquad (17)$$

$$b_t = \tau(S_t - S_{t-1}) + (1-\tau)(b_{t-1}) \qquad (18)$$

$$\overline{Z}_{t+1} = S_t + b_t \qquad (19)$$

where:

α:	Smoothing factor, where $0 \leq \alpha \leq 1$;
τ:	Trend factor, where $0 \leq \tau \leq 1$;
Z_t:	Known value of $F(x,t)$ at discrete time t;
S_t:	Expected value of $F(x,t)$ at discrete time t;
b_t:	Trend of $F(x,t)$ at discrete time t;
$\overline{Z}_{t+1}$:	Value of $F(x,t+1)$ predicted at discrete time t.

At each discrete time t, the VMPr Triggering method predicts next N values of $F(x,t)$ and triggers the VMPr phase in case $F(x,t)$ is predicted to consistently increase, considering that $F(x,t)$ is minimized.

3.1.4 Update-based Recovering for A1

When considering a two-phase optimization scheme for the VMP problem in cloud computing environments, the placement reconfiguration obtained in the VMPr phase is regarded as obsolete as time progresses during the algorithm running time due to its offline nature. That is why a new way of improving the placement taking into account the new requests is needed. The iVMP phase performs the recalculation of the improved placement. Consequently, the calculated new placement must be recovered according to the considered VMPr Recovering method before the reconfiguration is performed in operations.

The considered update-based VMPr Recovering method receives the placement reconfiguration calculated in the VMPr phase (corresponding to the discrete time $t-\beta$) and the current placement $x(t)$ as input data, as summarized in Algorithm **??**.

Considering that any VM V_j could be destroyed, or a cloud service could be scaled-in (horizontal elasticity) during the β discrete times where the calculation of the placement reconfiguration was performed, these destroyed VMs are removed from $x'(t-\beta)$ (step 1). Next, any resource from a VM V_j could be adjusted due to a scale-up or scale-down (vertical elasticity). Consequently, these resource adjustments are performed in $x'(t-\beta)$ (see step 2). Additionally, new VMs V_j could be created, or a cloud service could be scaled-out (horizontal elasticity), during the calculation of $x'(t-\beta)$. Finally, if the partially recalculated placement $x'(t-\beta)$ is better than the current placement $x(t)$, $x'(t-\beta)$ is accepted (step 5) and the corresponding management actions are performed (i.e. mainly migration of VMs between PMs). In case $x'(t-\beta)$ is not better than the current placement $x(t)$, no change is performed and the VMPr phase finishes without any further consequence.

3.2 Algorithm 2: Filter Scheduler

This work also evaluates the current default OpenStack Scheduler [**?**] for allocating VMs into PMs, identified as *A2*. This OpenStack VMP algorithm ($A2$) considers *filtering* and *weighting* for selecting a PM H_i to host a requested VM V_j for the considered iVMP phase.

For each requested VM V_j, the following set of filters are firstly applied to determine which PMs are eligible for allocating each requested VM:

- *RetryFilter*: if the PM H_i is available to host VMs. This is considered in the uncertain formulation with the binary variable $Y_i(t)$ that indicates if H_i is turned on $(Y_i(t)=1)$ or not $(Y_i(t)=0)$.
- *AvailabilityZoneFilter*: if the PM H_i is in the requested availability zone. The availability zone is mapped as a datacenter identifier c_i to fit in the considered uncertain formulation.
- *ComputeFilter*, *RamFilter*, *DiskFilter*: if the PM H_i has sufficient computational resources for allocating requested VM, as input data on $V(t)$.
- *ComputeCapabilitiesFilter*: to ensure satisfaction of additional specifications associated with the requested VM image. This is not considered in the uncertain formulation.
- *ImagePropertiesFilter*: to ensure that PM H_i has properties specified on the VM image. This is not considered in the uncertain formulation.
- *ServerGroupAntiAffinityFilter*: (if requested) to ensure that the requested VM will be allocated in a different PM than other VMs that compose the cloud service S_b.

Next, pre-selected PMs considering applied filters are then processed and weights are assigned to each PM, based on VM request specifications. Finally, PMs with the highest weight is selected and an incremental placement for the next time instant is returned. Table **??** summarize evaluated algorithms and methods.

Algorithm 4: Filter Scheduler in Algorithm $A2$.

```
Data: $H$, $V(t)$, $U(t)$, $x(t)$ (see notation in Section ??)
Result: Incremental Placement $x(t+1)$
foreach $V_j$ in $V(t)$ do
    $filtered-PMs$ = list of suitable PMs by applying
      filtering criteria
end foreach
foreach $V_j$ in $V(t)$ do
    $weighted-PMs$ = weight PMs from
      $filtered-PMs$
    select PM with the highest weight
end foreach
return Incremental Placement $x(t+1)$
```

4 Experimental Evaluation

The following sub-sections summarize the experimental environment as well as the main findings identified in the experiments performed as part of this work to validate the *Algorithm (A1)* proposed in [**?**] against the OpenStack Filter Scheduler, *Algorithm A2* (see Table **??**), considering scenario-based simulations with 400 different scenarios, taking into account average objective functions costs (see (**??**)).

4.1 Experimental Environment

The evaluated algorithms were implemented using Java programming language and considering the *Dynamic VMP Framework* available online[2]. Experiments were performed on a Windows 10 Operating System with an AMD A8-7410 APU with AMD Radium Graphics at 2.2 GHz CPU and 8 GB of RAM.

For more details on the considered experimental environment, as well as the 400 designed experimental workloads, interested readers may refer to [**?**].

4.2 Experimental Results

The main goal of the presented experimental evaluation is to validate that the previously proposed Algorithm *A1* [**?**] may result in a competitive implementation on an IaaS middleware such as OpenStack.

Table **??** presents values of the considered evaluation criteria, i.e. F_1 costs (see (**??**)), summarizing results obtained in performed simulations. The mentioned evaluation criteria are presented separately for each of the five considered IaaS cloud datacenter. It is worth noting that the considered IaaS cloud datacenters represent datacenters of different sizes and consequently, the considered workload traces represent different load of requested CPU resources (e.g. Low ($\leq 30\%$), Medium ($\leq 60\%$), High ($\leq 90\%$), Full ($\leq 98\%$) and Saturate ($\leq 120\%$)) workloads.

[2]http://github.com/DynamicVMP/dynamic-vmp-framework/releases

Table 1: Summary of evaluated algorithms as well as their corresponding VMPr Triggering and Recovering methods. N/A indicates a Not Applicable criterion.

Characteristics / Algorithm	**Decision**	**iVMP**	**VMPr**	**VMPr Triggering**	**VMPr Recovering**
A1 - inspired in [**?**]	Centralized	FFD	MA	Prediction-based	Update-based
A2 - inspired in [**?**]	N/A	Filter Scheduler	N/A	N/A	N/A

Table 2: Summary of evaluation criteria in experimental results for evaluated algorithms.

Criterion	Algorithm	Datacenter					
		DC_1	DC_2	DC_3	DC_4	DC_5	Ranking
F_1	**A1**	**0.752**	**0.838**	**0.926**	**0.934**	**0.983**	1st
	A2	0.794	0.932	0.986	1.003	1.019	2nd

Based on the information presented in Table **??**, it can be seen that Algorithm $A1$ outperformed Algorithm $A2$ in every experiment, taking into account the considered evaluation criterion (F_1). In summary, Algorithm $A1$ obtained better results (minimum cost) for considered evaluation criterion. When considering average objective function costs (F_1) as evaluation criterion, Algorithm $A1$ obtained between 4% and 11% better results than Algorithm $A2$.

5 Conclusions and Future Work

This work performed a first experimental evaluation of a previously proposed [**?**] two-phase optimization scheme for VMP problems in complex cloud computing environments, towards its implementation in a real-world IaaS middleware. For this, an industry de-facto standard as *OpenStack* was chosen and the *Filter Scheduler* was slightly adapted for simulations taking into account the considered VMP formulation.

The experimental evaluation presented in this work was mainly guided by previous work by some of the authors, considering that main contributions firstly proposed in [**?**] were taken into account to compare most promising studied algorithms ($A1$ in this case) against algorithms inspired in real-world ones (i.e. $A2$).

Experimental results demonstrate that the proposed algorithm $A1$ outperformed $A2$ in all considered experiments and may be considered as a promising algorithm for its implementation. Even do, several challenges still need to be faced in order address a good proposed tools for cloud computing datacenter management.

As a first step, IaaS middlewares such as *OpenNebula*, *vSphere Cloud* and other alternative tools with VMP algorithms should still be evaluated against Algorithm $A1$. This is proposed as future work.

Additionally, several assumptions should still be adapted to real-world situations or at least be evaluated under more scenarios, such as the recalculation time β that until now has been assumed to be a constant of discrete time instants. In real-world operations, this should be considered as a function of time t.

Several future works were also identified, mainly considering the novelty of the considered formulation. First, a formulation of a VMP problem considering a dynamic set of PMs $H(t)$, to consider PM crashes, maintenance or even deployment of new generation hardware is proposed as a future work.

Although modeling power consumption considering a linear relationship with CPU utilization is a very accepted approach in the specialized literature, considering the impact of other resources such as RAM and networking is proposed as future work.

Considering VMP formulations with more sophisticated cloud federation approaches is also left as a future work, taking into account the basic cloud federation approach considered in this work. Additionally, an experimental evaluation of alternative algorithms for both iVMP and VMPr phase is proposed as a future work, in order to explore performance issues with the proposed VMPr Triggering and Recovering methods.

Novel VMPr Triggering and VMPr Recovering methods could still be proposed to improve the considered two-phase optimization scheme in $A1$. The authors of this work also recognized the importance of jointly considering auto-scaling algorithms with the proposed two-phase optimization scheme for VMP problems, mainly for elastic cloud services as the considered in this work.

Experimenting with geo-distributed datacenters is also left as a future work, taking into account that simulations presented in this work considered only one cloud computing datacenter. Finally, fixed pricing is still very popular in cloud computing markets but emerging pricing schemes such as Spot Prices [**?**] should also be considered in real-world cloud computing datacenter operations.

6 Acknowledgements

This research is currently supported by CONACYT, in the context of the PINV15-781 "Software-defined Datacenters" research project grant.