Research Repository UCD

Title	Towards a Multi-Objective VM Reassignment for Large Decentralised Data Centres
Authors(s)	Saber, Takfarinas, Ventresque, Anthony, Brandic, Ivona, Thorburn, James, Murphy, Liam, B.E.
Publication date	2015-12-10
Publication information	Saber, Takfarinas, Anthony Ventresque, Ivona Brandic, James Thorburn, and Liam Murphy B.E. "Towards a Multi-Objective VM Reassignment for Large Decentralised Data Centres." Conference Publishing Services, 2015.
Conference details	2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, 7 - 10 December 2015
Publisher	Conference Publishing Services
Item record/more information	http://hdl.handle.net/10197/7312
Publisher's statement	© 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Publisher's version (DOI)	10.1109/UCC.2015.21

Downloaded 2023-04-07T10:52:29Z

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

Towards a Multi-Objective VM Reassignment for Large Decentralised Data Centres

Takfarinas Saber*, Anthony Ventresque*[†], Ivona Brandic[‡], James Thorburn[§], Liam Murphy*

*Lero@UCD, School of Computer Science and Informatics, University College Dublin, Ireland

Email: takfarinas.saber@ucdconnect.ie, {anthony.ventresque, liam.murphy}@ucd.ie

[†]IBM Research, Damastown Industrial Estate, Dublin, Ireland

[‡]Institute of Information Systems, Vienna University of Technology, Vienna, Austria

Email: ivona@ec.tuwien.ac.at

[§]IBM Software Group, Toronto, Canada

Email: jthorbur@ca.ibm.com

Abstract—Optimising the IT infrastructure of large, often geographically distributed, organisations goes beyond the classical virtual machine reassignment problem, for two reasons: (i) the data centres of these organisations are composed of a number of hosting departments which have different preferences on what to host and where to host it; (ii) the top-level managers in these data centres make complex decisions and need to manipulate possible solutions favouring different objectives to find the right balance. This challenge has not yet been comprehensively addressed in the literature and in this paper we demonstrate that a multi-objective VM reassignment is feasible for large decentralised data centres. We show on a realistic data set that our solution outperforms other classical multi-objective algorithms for VM reassignment in terms of quantity of solutions (by about 15% on average) and quality of the solutions set (by over 6% on average).

Keywords—Large Decentralised Data centres, Machine Reassignment, Multi-objective Optimisation, Evolutionary Algorithms.

I. INTRODUCTION

Background

Virtual Machine (VM) placement is known to be one of the biggest challenges in data centres management [1], as tackling server underutilisation is desirable [2] but extremely complex [3] given the constraints and the scale of the problem. Various definitions of this VM placement problem have been proposed in the past years, addressing various facets, such as, initial assignment [4], reassignment based on some objective (e.g., number of service level agreements violations [5] or electricity consumption [6]), physical machines (PM) and workload heterogeneity [7], [8] or hot-spot mitigation [9], [10].

Recently, *multi-objective VM reassignment* has been described as an important research challenge for data centre optimisation [1] and various solutions have been proposed [11], [12]. However, they often consider only small scale systems and use a ‘weak’ definition of multi-objective: weighted sum of objectives [13] or fine-tuned bi-objective resolution [14].

The *brokering problem* is a relatively new challenge, where data centres are considered composed of several entities, exchanging VMs, often under the supervision of a middle entity (broker), to optimise costs and SLAs [15], [16]. This problem is also referred to as *open cloud*, or *multi-clouds*, management. This is a relevant problem for the industry, as companies’

IT infrastructures are nowadays hosted in various locations. However, most of the solutions proposed in the literature for this problem do not consider ‘competition’ between hosting sites, while it is always possible in practice that hosting sites have different objectives, and they assume an a priori cost of placing a VM on a PM, which we think is not always possible (especially if the hosting sites have some sort of autonomy).

Research Challenge

Large modern organisations often face internal segmentation, based on geography, legal jurisdictions, age/maturity of lines of business, and sheer size. This process, sometimes referred to as siloing, means that the organisation has employees who identify themselves with their group rather than with the whole organisation, and who view their group’s objectives as being more relevant than the organisation’s objectives. Hence, capital allocators (CA) of the different hosting departments sometimes compete or at least have different perspectives on the best way of making the system better. It is crucial for large organisations to address the difference in objectives and incentives between capital allocators, and in many large organisations it is seen as one of the biggest barriers to optimisation.

Similarly, the notion of *best* reassignment is controversial at the top level of organisations, as it is more about the right balance between the different objectives that will be favoured. Decision makers in large organisations tend to favour tools that allow them to manipulate *good reassignments*, i.e., possible solutions that are better than every other possible reassignment on a particular combination of objectives. Given these good reassignments, they can use *policies* (i.e., rules regarding VM reassignments) to limit the possible reassignments to those meeting their preferences. Eventually, decision makers can look at the different possible reassignments and make a decision based on local optimisation (e.g., favouring objective 1 which gives a bigger gain than objective 2 while the latter is usually more important).

Hence the research challenge we address in this paper is the possibility to build a system that: (i) satisfies individual CAs’ placement preferences, (ii) offers top level decision makers with a panel of possible and validated good assignments that they can navigate and (iii) scales up to large data centres.

Figure 1: Possible reassignments in 2-dimensions (in red, the good ones) and area defined by an organisation’s policies (in grey).

Example

As a motivating example, consider an organisation with various hosting departments, each of them managed by capital allocators with different placement preferences. For instance some CAs may consider that energy consumption is the most important element, while for others it can be the cost of licensing; or some CAs see the reliability as the key element (for instance if they run critical applications), while other CAs have a strict policy regarding response time and then collocation of VMs. Given these preferences, the organisation can try to find reassignment solutions that satisfy the constraints of the IT infrastructure and the CAs’ preferences. Figure 1 shows a reassignment problem with two objectives: the electricity consumption and the reliability (the lower the better for both). The good reassignments, i.e., those that are better than any other in a particular combination of objectives, are in red. Solution e, for instance, is not a good reassignment as it is worse than b on both objectives - same applies to f wrt. c. The organisation may define some policies, i.e., some rules that describe the desired optimisations: here a maximal value for the electricity consumption and the reliability, defining the area of good assignments that the decision makers can select (grey area on Figure 1). There are still two possible reassignments, b and c. The decision makers can then evaluate them locally and make a relative decision among them, such as “c gives me a better electricity cost but it is not a huge gain, while b’s reliability is higher – so we favour b”.

E-GeNePi

In this paper, we propose *E-GeNePi*, a multi-objective VM reassignment system for large decentralised data centres. Such data centres are composed of various autonomous hosting departments, or *virtual data centres (VCs)*, with their own preferences (i.e., more formally, a weighted sum of the objectives). At the top level of those large data centres stand managers who make optimisation decisions based on a set of good, possible, reassignments. *E-GeNePi* makes *assignments* of VMs to hosting departments at the top level of the organisation and each hosting department makes *placement* of the VMs on their PMs based on the assignments and their preferences. Eventually *E-GeNePi*’s output is a list of solutions that are better than any other in a particular combination of objectives. Our contributions in this paper are:

- We provide the first *full formalisation of the multi-objective VM reassignment problem for large decentralised data centres*.
- We propose a novel architecture composed of two modules, a multi-objective assignment module for the decision makers and an optimisation placement module for the VCs. We propose an implementation for the two modules called *E-GeNePi*, with a three step metaheuristic at the top level and a composition of first fit and hill climbing at the individual VC level.
- We perform a *thorough evaluation* of *E-GeNePi* using a realistic data set and we compare it to solutions where the reassignment module is replaced in *E-GeNePi* with other state-of-the-art multi-objective algorithms. We show that *E-GeNePi* outperforms all of them in terms of number of found non-dominated solutions (+14.84% than second best on average) and variety of these solutions (+6.03% hypervolume than second best).

In this paper, we use the phrase *data centre* for the global IT infrastructure of a large company, while we use *virtual data centre (VC)*, or hosting site/department, for each smaller group of servers and VMs that make sense by themselves. For instance, a company newly acquired by a bigger one and keeping its servers to host its workloads would be considered as a VC. Likewise, a particular department in a big group ‘possessing’ its own IT infrastructure would be considered a VC. We describe as *decentralised* the type of data centres that our system targets, as they are composed of several VCs with a certain degree of autonomy in the placements they favour.

We also use ‘objectives’ and ‘preferences’ in this paper and we would like to clarify the distinction. The *objectives* are the different dimensions that need to be optimised. For instance, we define in the next section three objectives that make sense for practitioners when optimising a data centre (see next sections for more details): electricity consumption, system reliability and migration. These objectives are agnostic towards any particular goal or target: they just define the multi-objective search space of all possible assignments. The *preferences* on the contrary define a direction in the multi-objective space, in the form of a set of values, or weights, one for each objective. For instance CAs aiming at minimising the electricity consumption of their VCs at the expense of the other objectives would have a preference noted (1, 0, 0) (assuming electricity cost is the first element of the triplet). Managers at the top level of the data centre are more keen on seeing the different placement options before making a decision – as we explain above, this is a complex process which may require discussions and evaluation of the cost/benefit of particular plans.

In general, we say that reassignments are possible if they satisfy all the constraints of the VM placement problem, and good if they belong to the *Pareto front*, i.e., the set of placements that are better than any other in one particular combination of the objectives. For instance, in Figure 1, reassignment b is not as good as a for the reliability objective, or c and d for electricity consumption; but none of the other reassignments (including a, c and d) is better than b in both objectives. In general, for a NP-hard problem such as the VM placement problem, especially given the size of the instance

we work with, the exact Pareto set is unattainable and what we call Pareto set is the set of good placements at a particular iteration – typically, when a time limit is reached or no more improvement is obtained.

The rest of this paper is organised as follows: Section II proposes a formal problem definition (i.e., the constraints and the objectives); Section III presents E-GeNePi; Sections IV and V give a description of the experimental setup and present the evaluations. Eventually, Section VI concludes this work.

II. FORMAL PROBLEM DEFINITION

We propose in this section the first full formalisation of the multi-objective VM reassignment problem in large decentralised data centres. We present the variables defining a data centre: physical machines (PMs, i.e., servers), virtual machines (VMs), individual virtual data centres (VCs); as well as the constraints, the cost functions and the optimisation problems: placement and reassignment. We refer the reader to [17], [18] for more details on some of the notations and concepts. [17] is a recent definition of the VM placement problem proposed by Google for an Operation Research challenge. We explain in Section IV why it provides a well regarded formalisation of the VM placement problem (not a multi-objective one though) and why it is relevant for the industry as well. [18] is a multi-objective extension and a linear formulation of the Google VM placement problem for centralised data centres.

A data centre is composed of a set C of VCs: $C = \{c_1, \dots, c_n\}$. Each c_i is given a set $\mathcal{M}_i \subseteq \mathcal{M}$ of PMs, $\mathcal{M}_i = \{m_1, \dots, m_n\}$. Each m_j has several resources $r \in \mathcal{R}$ (e.g., RAM, CPU, disk), in limited capacity $Q_{m_j,r}$. Resources are either transient ($r \in \mathcal{TR} \subseteq \mathcal{R}$, such as RAM and disk) if they are consumed by both origin and destination machines during a VM migration, or non-transient otherwise (e.g., CPU): $r \in \mathcal{R} \setminus \mathcal{TR}$. The data centre hosts a set $\mathcal{V}$ of VMs, $\mathcal{V} = \{v_1, \dots, v_l\}$: $M(v_k) = m_j$ is used to denote the PM on which v_k is hosted (we also use the notation $M_0(v_k) = m_j$ for the original PM of v_k in case of a migration). The quantity of resource r that every v_k needs is fixed to $d_{k,r}$. VMs are sometimes grouped into services $\mathcal{S} = \{s_1, \dots, s_p\}$, with $s_p = \{v_p^1, \dots, v_p^q\}$. A service can be seen as a distributed application, such as one implementing a multi-tier architecture.

A. Constraints of the problem

Every placement of VM on a PM is subject to a set of constraints that we describe below.

1) *Capacity constraints*: require that VMs do not exceed the resource capacity of the PM they are hosted on or they are migrating to/from (in case of transient resources), $\forall m \in \mathcal{M}$:

$$\begin{cases} \sum_{v_k \in \mathcal{V} | M_0(v_k)=m_j \vee M(v_k)=m_j} d_{v_k,r} \leq Q_{m_j,r} & \text{if } r \in \mathcal{TR} \\ \sum_{v_k \in \mathcal{V} | M(v_k)=m_j} d_{v_k,r} \leq Q_{m_j,r} & \text{otherwise} \end{cases} \quad (1)$$

2) *Anti-cohabitation constraints*: reflect the fact that services running on several VMs may prefer to use a different PM for each VM.

$$\forall v_k, v_l \in \mathcal{V}, k \neq l, \forall s_p \in \mathcal{S}, (v_k, v_l) \in s_p^2 \Rightarrow M(v_k) \neq M(v_l) \quad (2)$$

Network connections between PMs in data centres vary and this can have an impact on CAs' decisions on the location of VMs. The concept of neighbourhood is used to capture this and we denote it $N(m_i) = \{m_i^1, \dots, m_i^k\}$ for a PM m_i .

3) *Dependence constraints*: are used when services depend on one another: if a service s_i depends on s_j ($s_i \hookrightarrow s_j$), then every VM v_k in s_i should be in the neighbourhood of at least one VM v_l from s_j :

$$s_i \hookrightarrow s_j \Rightarrow \forall v_k \in s_i, \exists v_l \in s_j \mid N(M(v_k)) = N(M(v_l)) \quad (3)$$

4) *Spread constraints*: express that some services require their VMs to be allocated to different VCs, for instance for security reasons:

$$\sum_{c_i \in C} \min(1, |\{v \mid v \in s \wedge VC(v) = c_i\}|) \geq spread_s, \quad \forall s \in \mathcal{S} \quad (4)$$

where $spread_s$ is the minimum number of VCs that have to contain at least one VM from the service s , and $VC(v)$ is the VC where the VM v is located.

We give a visual representation of the constraints and their effect on PMS, VMS and VCs, in Figure 2. This little data centre is composed of 2 VCs, 5 PMs, 10 VMs and 3 services. The capacity constraints are not presented here but we assume that the placements shown in Figure 2 satisfy them. The anti-cohabitation constraints are shown by the fact that no two VMs from the same service are on the same PM: for instance, VM1, VM2 and VM3 from S1 are assigned to 3 different PMs. The dependence constraints (round headed arrows in Figure 2) require for all VMs of S2 to be in the neighbourhood of VMs of S1 (e.g., VM4 $\in$ S2 is co-located with VM2 $\in$ S1) or S3. The spread constraint (2), in this scenario with 2 VCs, means that every service needs to spread its VMs over the two VCs.

B. Objectives to optimise

We now introduce the cost functions defining the different objectives.

1) *Reliability Cost*: Very often VM placement algorithms assume a buffer of resources that they reserve, as safety capacity [10]. That allows them not to consider the potential issues with the co-location of VMs – when several VMs are hosted on the same server they may generate contention on some resources (e.g., cache). More recently, researchers [19] have proposed to adapt the buffer's size to the hosted VMs' workloads: given the pattern of utilisation of the resources by the VMs, the authors estimate the risk of resource violation and place together VMs that are unlikely to use shared resources at the same time. As it is easier to understand and this is not the main focus here, in this paper we follow the classical direction and use a fixed safety capacity as a metric for the *reliability cost*, i.e., the likelihood of resource contentions for a server.

A machine m_j is said reliable if its load for every resource $r \in \mathcal{R}$ does not exceed a safety capacity $\rho(m_j, r)$, and we define the *reliability cost* associated as:

$$\rho(m_j) = \sum_{r \in \mathcal{R}} \max(0, Q_{m_j,r} - \rho(m_j, r)) \quad (5)$$

Figure 2: Example of a feasible assignment of VMs to VCs and their placement on PMs (spread= 2).

2) *Cost of migrating*: The cost of migrating a VM v is the composition of three costs [20]: preparation of the migration: $\mu_1(v, M_0(v))$, transmission: $\mu_2(v, M_0(v), M(v))$ and installation in the new host: $\mu_3(v, M(v))$. Note that each of these costs depends on various parameters, such as, VM size and network topology.

$$\mu(v, M_0(v), M(v)) = \mu_1(v, M_0(v)) + \mu_2(v, M_0(v), M(v)) + \mu_3(v, M(v)) \quad (6)$$

3) *Electricity Cost*: of a machine $m_j \in \mathcal{M}$ is defined by the electricity price (per unit) at m_j 's location multiplied by m_j 's electricity consumption. Modelling electricity consumption is complex and we use here some simplified model [21], [22] considering that it is a linear function of the CPU usage:

$$\epsilon(m_j) \begin{cases} \gamma_{m_j} \times (\alpha_{m_j} \times Q_{m_j, CPU} + \beta_{m_j}) & \text{if } m_j \text{ is running} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

where γ_{m_j} is the electricity price at the location of m_j , β_{m_j} its electricity consumption when idle, and α_{m_j} the tangent of the CPU consumption.

Definition 1 (Placement): Given a VC c_i , its PMs $\mathcal{M}_i$ and a set of VMs $\mathcal{V}_i = \{v_1^i, \dots, v_2^i\}$, a *placement* of $\mathcal{V}_i$ on $\mathcal{M}_i$ is a mapping: $Plac_i : \mathcal{V}_i \mapsto \mathcal{M}_i$, such that $Plac_i(v) \rightarrow m$, which satisfies the constraints 1, 2, 3.

The preference of each VC c_i corresponds to a function to minimise and is defined by three constants w_i^1 , w_i^2 and w_i^3 : $w_i^1 \times \sum_{m_j \in \mathcal{M}_i} \rho(m_j) + w_i^2 \times \sum_{v_k \in \mathcal{V}_i} \mu(v_k, M_0(v_k), Plac(v, \mathcal{M}_i)) + w_i^3 \times \sum_{m_j \in \mathcal{M}_i} \epsilon(m_j)$.

Definition 2 (Reassignment): A reassignment ReA of VMs to VCs is a mapping: $ReA : \mathcal{V} \mapsto \mathcal{C}$, such that $ReA(v_k) \rightarrow c_i$, which satisfies the constraint 4 and the constraints 1, 2 and 3 at c_i 's level.

As we are in a multi-objective context, we do not try to minimise one single objective function but rather all of

Figure 3: Overview of E-GeNePi

the objectives (defined as minimisation of the various cost functions 5, 6 and 7) at the same time. The solutions define what is called a *Pareto front*, i.e., a set of solutions that are better than any other on a particular combination of objectives, and are not comparable to any other on the Pareto front.

III. E-GeNePi

E-GeNePi is our multi-objective VM reassignment system for large decentralised distributed data centres. The goal of E-GeNePi is to satisfy the preferences of the individual VC's CAs while offering the top managers with better placement solutions in the multi-objective search space. E-GeNePi is composed of two different modules (see Figure 3):

Assignment: The decision makers at the organisation level run GeNePi [23], an hybrid metaheuristic that successively applies three heuristics to (i) find good initial solutions using a greedy algorithm (GRASP), (ii) find solutions scattered in

the search space to increase the variety of solutions (using a genetic algorithm: NSGA-II) and (iii) increase the number of solutions using a local search (PLS). Decision makers send every reassignment solution found by GeNePi to the VCs, where a placement is tried: if a placement is possible for every VC, then the real cost of the solution is updated (and the solution possibly discarded if not good enough); otherwise GeNePi modifies the solution and resubmits it. The first step of GeNePi corresponds to the constructive phase of GRASP [24]: VMs are sorted according to their dependencies and requirements (decreasingly) and the top ones are assigned iteratively to VCs that respect services spread, allow the satisfaction of dependency constraints and have a utility higher than a given threshold α . GeNePi then applies a specific genetic algorithm NSGA-II [25] where: (i) four parents are selected instead of two during the evolution phase and a tournament is organised by pairs, before mixing the winners (using crossover and mutation operators); (ii) the evolved population is mixed with the original one in order to only keep the elites and quicken the convergence towards non-dominated solutions. To be more precise, the crossover in our system is performed by one or two ‘cuts’ at random positions in a solution’s parents chromosomes and the resulting parts are interchanged, while the mutation consists in reassigning a random VM to a random VC.

Eventually GeNePi uses a local search, PLS [26], which looks for new solutions in the neighbourhood of the most isolated non-dominated solutions (as they are the most likely to refine the non-dominated set). GeNePi uses two operators in the generation of the neighbourhood: 1-exchange and swap.

We define an execution time for each step instead of fixing the number of iterations. This allows to have more control over the global execution time and to adapt to the complexity of the instances (e.g., an iteration of GRASP is faster when the constructed solution is infeasible). GRASP is given a third of the execution time, and that can be extended up to the half if the size of the initial population for NSGA-II is not reached. NSGA-II gets half of the global time, because of the large improvement it can achieve. PLS gets a sixth of the global time as it is only used to refine the non-dominated set.

Placement: each VC runs two algorithms: (i) a placement algorithm to match allocated VMs to hosted PMs; and (ii) a hill climbing algorithm to optimise the placement. If the two algorithms are successful (i.e., the output is a feasible solution), CAs send the cost of placing the set of allocated VMs back to the decision makers. Otherwise, if no placement is possible, the VMs are sent back to the managers of the organisation. The placement process has to be done in less than a certain time (the same for all the VCs).

The first step is described in Algorithm 1 and consists in finding an initial placement satisfying every VC’s constraints. This is done by placing VMs already in the VC at their initial PM, thus minimising the migration cost. The rest of the VMs are sorted by dependencies and requirements, and placed one after the other using a First-Fit Decreasing. The reparation mechanism deals with VMs that cannot find a placement and moves at most two other VMs; or cancel the placement. The second step is summarised in Algorithm 2: it uses a hill climbing algorithm with two possible terminations: either when there is no more improvement of the objective function or when the time-out (set by the system) is reached. Similarly

---

Algorithm 1: Find initial placement

```

input :  $M_i$ : List < Machines >,  $V_{allocated}$ :
         List < VMs >
output: unfitted: List < VMs >
 $unfitted \leftarrow \emptyset$ ;  $placed \leftarrow \emptyset$ ;  $newV_i \leftarrow \emptyset$ ;
// VMs already in the VC are not moved
 $newV_i \leftarrow V_{allocated} \setminus V_i$ ;
// Placement of VMs that were not in the VC
sortOnDependencyAndRequirements( $newV_i$ );
for  $v \in newV_i$  do
  if exists(firstFit( $v$ )) then
     $M(v) \leftarrow firstFit(v)$ ;  $placed \leftarrow placed \cup \{v\}$ ;
  else
     $unfitted \leftarrow unfitted \cup \{v\}$ ;
// Reparation for the VMs that cannot be placed
for  $v \in unfitted$  do
  if isRepaired( $v, M_i, placed$ ) then
     $unfitted \leftarrow unfitted \setminus \{v\}$ ;
return unfitted;

```

---

to PLS for the assignment, the idea is to find a better placement by exploring the possible solutions around the current feasible one, using 1-exchange (which changes the placement of one single VM) and swap (which exchanges the placement of two VMs).

Algorithm 2: Hill climbing algorithm in a VC

```

input :  $s$ : Solution, Weights: Array < real >,
         Moves: List < Move >,  $T$ : Time
output:  $s$ : Solution
 $cost \leftarrow getCost(s, Weights)$ ;
 $N \leftarrow generateNeighbours(s, Moves)$ ;
(saving, neighbour)  $\leftarrow hasMaxSaving(s, N)$ ;
if saving > 0 & execTime <  $T$  then
   $s \leftarrow Hill\_Climbing(neighbour, Weights, Moves, T)$ ;
return  $s$ ;

```

IV. EXPERIMENTAL SETUP

We want in this section to evaluate how E-GeNePi performs against state-of-the-art multi-objective placement metaheuristics. We use a large data set provided by Google and two metrics: a measure of the quantity of solutions found and a measure of the quality of the solutions (how much of the solution space they explore). All our algorithms are implemented in C++ and the tests are performed on a machine running Ubuntu 12.4 LTS 64bits with 62GB of RAM and 24 Intel(R) Xeon(R) 2.20GHz CPUs.

A. Data set

The Google ROADEF challenge data set [17] consists of a large number of realistic instances (each representing one data centre) with their VMs, PMs, resources, services, VCs (or ‘locations’). This data set does not provide a multi-objective formulation though and we had to adapt the instances

Instance	# Resources	# PMs	# VMs	# Services	# VCs	Execution time of E-GeNePi (s)
a_1_1	2	4	100	79	4	15
a_1_2	4	100	1,000	980	4	1,200
a_1_3	3	100	1,000	216	25	1,200
a_1_4	3	50	1,000	142	50	1,200
a_1_5	4	12	1,000	981	4	1,200
a_2_2	12	100	1,000	170	25	3,600
a_2_3	12	100	1,000	129	25	3,600
a_2_4	12	50	1,000	180	25	3,600
a_2_5	12	50	1,000	153	25	3,600
b_2	12	100	5,000	2,462	10	7,200
b_3	6	100	20,000	15,025	10	7,200
b_4	6	500	20,000	1,732	50	7,200
b_5	6	500	40,000	35,082	10	7,200

Table I: Instances size and allowed execution time.

(note that the participants of the challenge optimised only one single weighted sum of the costs proposed – hence there is no possible comparison between our work and others using ROADEF). Our instances aim to model realistic scenarios as we observe them in large companies.

The only things we had to add to the challenge are the electricity cost variables α_{m_j} , β_{m_j} and γ_{m_j} for each machine m_j , and the preference of each VC towards the objectives: w_i^1 , w_i^2 and w_i^3 for each VC c_i (see Section II). We focus here on 14 instances (see Table I): all the instances of the sets a_1 (easy), 4 (out of 5) of the set a_2 (medium), and the first 5 (out of 10) of the set b (hard). We picked these instances only to limit the execution time of our experiments (we have 10 runs per algorithm and per instance). Instances from the set a_1 have between 2 and 4 resources, 4 to 100 PMs, 100 to 1,000 VMs, 79 to 981 services and 4 to 50 VCs; those from the set a_2 have 12 resources, 50 or 100 PMs, 1,000 VMs, 129 to 180 services and 25 VCs; while those from the set b have 6 or 12 resources, 100 or 500 PMs, 5,000 to 40,000 VMs, 1,732 to 35,082 services and 10 to 50 VCs. For example, the instance a_1_2 that we use in the next section to showcase the behaviour of the algorithms has 4 resources, 100 PMS, 1,000 VMs in 980 services and 4 VCs. E-GeNePi is allowed only a fixed execution time (and we force all other algorithms to give an answer in this limited time): 15 seconds for a_1_1, 1,200 seconds for the other instances of the set a_1, 3,600 seconds for the instances of the set a_2 and 7,200 seconds for the instances of the set b. These time limits are considered realistic in the context of our work (optimisation performed on a regular basis, e.g., monthly or quarterly).

B. Metrics

Comparing the quality of two multi-objective algorithms is difficult [27], which explains why several metrics are in use such as, coverage, diversity, number of non-dominated solutions [28]. We wanted in our work to compare the different algorithms from two perspectives:

Quantity: of solutions found: this is an intuitive yet important measure as we want to provide the decision makers with multiple options that can be navigated until they find one to implement. We use the *size of the set of non-dominated solutions*, i.e., the number of solutions on the Pareto front (those that are better than any other on a particular combination of

Figure 4: Metrics: number of Pareto solutions (red dots) and hypervolume (grey area).

objectives). Figure 4 shows a two-dimensional reassignment, with the non-dominated solutions in red and the other (not interesting) solutions in light blue colour. The initial placement is one of the solutions, generally not on the Pareto front.

Quality: of the set of solutions: the more of the search space is covered by the set of solutions the better it is for the decision makers who can then explore the variety of solutions and pick the one that corresponds the best to their needs. We use the *hypervolume* (or S-metric) [29] as a measure of the quality of a set of solutions. The hypervolume is now popular [30] in the optimisation community to compare non-dominated sets obtained by different multi-objective algorithms. It measures the dominated hypervolume of space between the non-dominated solutions and a reference point selected far from it. This reference point must be the same for all the algorithms, and differs only between two instances. Figure 4 shows the hypervolume (grey area) computed from a reference point and the set of non-dominated solutions.

Whenever we need to compare two placements or assignments we will also use the measure of *savings*, which is the difference between the costs of two placements and reflects the improvement made (if the difference is positive).

C. Other Algorithms

We compare our solution *E-GeNePi* to five other algorithms that apply the same process at the VC level (find an initial placement and apply a hill climbing algorithm). They differ from each other on the multi-objective algorithm that generates the assignments (at the capital allocator level). The first algorithm is *E-GRASP*, where GeNePi is replaced by GRASP. The second algorithm is *E-NSGA-II* that runs NSGA-II. The third algorithm (*E-GrGA*) has a two steps method with GRASP (one third of the execution time) followed by NSGA-II (two thirds of the execution time). *E-PLS* is the fourth algorithm, where the decision maker runs PLS during the entire execution, but applied on all the non-dominated solutions at every step. The last algorithm is *E-HC*: it runs several iterations of the *hill climbing* algorithm with different weight triplets (one weight per objective).

V. EVALUATION

The main goal of this section is to compare the quantity and the quality of the solutions found by E-GeNePi against the other algorithms that we presented above. We are also interested in evaluating the hill climbing algorithms at the VCs level, as it is a crucial for E-GeNePi to know whether the VCs find a placement for the VMs assigned to them in a reasonable time (i.e., the quicker the better). Finally, we are interested to know how the different algorithms work together in E-GeNePi, and we profile the resolution of one of the instances to understand the behaviour of the algorithms.

A. Evaluation of the VCs' Hill Climbing

At the VC level (see Figure 3, Section III) we use both a first fit algorithm for the initial placement of VMs and a hill climbing to improve the placement. To understand how the hill climbing algorithm improves the placement, and how much time is needed to reach a stable state (when the algorithm converges), we show on Figure 5 the iterative savings of the instances with a sufficient number of PMs per VC (i.e., those where number of PMs $\gg$ number of VCs) – we use the very first assignment found by E-GeNePi here. We first show all the details of the process for the 4 VCs of instance a_1_2 , each composed of 25 PMs and 225, 228, 284 and 233 VMs respectively. The first thing to say is that we do not see here the first placement (first fit decreasing step) but only the improvements of the placements allowed by the hill climbing. We notice that the savings are significant and decreasing. This is interesting for us as it means that CAs (or decision makers) can decide to allocate a shorter time for the improvement of the placement without impacting too much the quality of the process. Figure 5 also shows that for a_1_2 the placements quickly converge (sometimes locally) to an optimum: in our example all the placements converge before the end of the allocated time (4 seconds) – which may not always be the case. We also see that the original situation varies a lot and the amount of savings as well: during about the same execution time (3.64 and 2.95 seconds respectively) savings for c_2 and c_0 vary a lot (6,081,944 for c_2 while 2,192,652 for c_0). Likewise, c_2 needs more time to converge than the other VCs. Both remarks are caused by the higher number of VMs on c_2 than other VCs which makes improving the placement more complex (longer) while it exhibits more important savings.

Figure 5 also shows a simplified version of the savings for each of the other instances. Instead of showing the savings for every instance, and because most instances have a lot of VCs (e.g., 10 to 50 in most cases), we present only the VCs that converge the fastest (i.e., the first to have no improvement between any two iterations) and the slowest, and the mean of all VCs' savings values. We observe that the convergence is very quick, less or around 1 second for all the a's (but a_1_2), and in the order of the tens of seconds for the b's. We also see that the difference between fastest and slowest VCs is not huge, and anyway in the same order of magnitude (19.5s-60.42s for b_1 and 5.5s-19.93s for b_4 are the largest differences between fastest and slowest converging VCs). The mean values (dark lines in the grey areas) show that the converging time of the majority of the VCs tend to be close to the one of the slowest. It is not a problem as such, given that processing the VCs is likely to be done in parallel, i.e., at each VC's and that even for the slowest the placement improvements converge in a reasonable time.

B. Quantity and Quality of Solutions

Table II shows that using E-GeNePi gives better results in terms of hypervolume (quality of solutions sets) for all instances, with an average of +6.03% improvement compared to the second best results. This means that E-GeNePi succeeds in finding new areas of the search space and exploring them. While E-GrGA, E-GRASP and E-NSGA give good results on average, E-NSGA is bad when the instances are big. This can be explained by the fact that NSGA is dependent on several initial solutions of good quality – and we have only one here (the initial placement), requiring to generate some random (and of poor quality) ones. Algorithms based on a local search (i.e., E-PLS and E-HC) have the worst results, which is due to the time it takes to evaluate an entire neighbourhood in absence of utility functions (the algorithms need to ask the VCs).

Table II also shows that E-GeNePi finds the highest number of solutions (10 out of 14 instances) or the second highest (4 remaining instances): on average, E-GeNePi finds 14.84% more solutions than the second best or best algorithms. We also notice a large improvement from the two step method (GrGA) with an increase of 229.91% in the number of found solutions (and 41.96% without including a_1_1). GrGA, GRASP and NSGA have fluctuating results depending on the instance, and PLS does not perform well. The new thing we learn is that HC gets a fair number of solutions for easy and medium instances (they take less time to evaluate), but gets worse results for others.

C. Profiling

We plot on Figure 6 the evolution of the hypervolume over time for the different algorithms on the a's instances. The first thing to notice is that algorithms based on GRASP (E-GeNePi, E-GrGA and E-GRASP) are aggressive at the start, due to the greedy nature of GRASP and the usage of weights that lead to a large spread over the research space. After 10-20% of the time allocated, E-GRASP though plateau. We also see that E-GeNePi and E-GrGA have almost the same behaviour until a stage when NSGA-II reaches an elite population (no more improvements) while E-GeNePi increases the number of solutions thanks to its last step, PLS. E-GRASP mixes

Figure 5: Placement savings obtained by the Hill Climbing algorithm: detailed for a_1_2 (4 VCs), and, for the others, area defined by the fastest and slowest VCs (in terms of convergence), and mean of the savings.

randomness and aims to get the best possible improvement, which leads it occasionally to have a good improvement (e.g., between 862s and 1,286s in a_1_2) when it finds randomly a good solution; but it can also end up stuck for a long time. E-NSGA shows it can improve the hypervolume, but this is

spoiled by the first phase, i.e., the generation of a population from the initial assignment which looks bad. This issue is less dramatic when the search space is small (i.e., for the small instances). Then the random generation does not seem too bad – as the hypervolume increases well and solutions are found.

	E-GeNePi	E-GrGA	E-GRASP	E-NSGA	E-PLS	E-HC
a_1_1	416 (1.5)	15 (1.01)	25 (1.14)	20 (1.13)	6 (1.0)	276 (<i>1.49</i>)
a_1_2	36 (12.66)	26 (<i>11.55</i>)	22 (9.9)	37 (10.61)	9 (6.21)	14 (8.74)
a_1_3	74 (8.14)	40 (7.23)	23 (6.31)	44 (7.27)	3 (1.05)	45 (5.96)
a_1_4	192 (42.39)	163 (4.13)	11 (2.41)	<i>189</i> (4.22)	42 (1.89)	72 (3.35)
a_1_5	39 (20.45)	34 (18.35)	17 (17.62)	20 (20.30)	6 (9.99)	25 (16.95)
a_2_2	62 (20.09)	46 (16.75)	23 (9.30)	55 (<i>17.61</i>)	6 (6.26)	65 (13.85)
a_2_3	74 (19.55)	39 (<i>18.13</i>)	18 (8.44)	72 (18.00)	4 (3.82)	22 (4.46)
a_2_4	118 (43.13)	92 (<i>36.03</i>)	27 (5.73)	<i>113</i> (33.47)	3 (0.93)	51 (6.42)
a_2_5	67 (4.49)	95 (4.39)	19 (1.78)	59 (3.97)	8 (0.22)	64 (1.14)
b_1	17 (10.96)	14 (<i>10.60</i>)	18 (10.26)	18 (3.72)	3 (3.72)	5 (10.09)
b_2	17 (12.32)	14 (11.93)	13 (<i>11.99</i>)	12 (5.62)	3 (5.63)	5 (7.57)
b_3	7 (5.54)	6 (5.37)	4 (5.31)	3 (1.51)	3 (1.51)	2 (1.71)
b_4	16 (12.19)	6 (<i>11.50</i>)	8 (10.84)	<i>11</i> (4.27)	12 (4.27)	2 (4.29)
b_5	7 (6.68)	5 (6.57)	5 (6.51)	1 (4.18)	2 (4.18)	1 (4.18)

Table II: Summary of solutions found and hypervolume (in brackets) for the various algorithms and the various instances. For both metrics, the higher the better. We put in bold the best values for each instance, and in italics the second best.

Figure 6: Evolution of the hypervolume during the execution of the a's instances for the different algorithms. Note that each dot corresponds to a solution (or group of solutions) found.

Because they require a long evaluation of the neighbourhood of a solution, algorithms based on a local search (E-PLS and E-HC) perform poorly. E-HC is relatively better though, especially for the smaller instances.

VI. CONCLUSION

This paper gives the first full formal definition of the multi-objective VM reassignment problem for large decentralised data centres. This problem is deemed important for many large organisations, where hosting departments have a certain degree of autonomy and the capital allocators' preferences are not always aligned with the main objectives of the organisation. This problem is challenging given the size of real world IT infrastructures, the variety of individual capital allocators' preferences and the complexity of the multi-objective resolution.

We have proposed E-GeNePi, a solution composed of two modules: one for the capital allocators of the various hosting departments (VCs) which aims at finding a placement of VMs on the VC's PMs and improving this placement as much as they can (using a first fit decreasing algorithm and a hill climbing algorithm); and one for the decision makers at the top of the organisation, which aims at finding a broad set of good reassignments (i.e., Pareto, with no other solution that beats them on every objective). A comparison on a realistic data set showed that E-GeNePi outperforms other systems implementing multi-objective algorithms in terms of quantity of solutions (+14.84% on average) and quality of the solutions set (+6.03% on average).

The focus of our work will now be on the interactions between VCs and the organisation, and between hosting departments themselves. The autonomy of the VCs means that they should be able to discard assignments given to them more easily, for instance if they do not meet their preferences, or to enforce some placements (e.g., if they want some specific VMs). In this context, an organisation needs to incentivise the assignments to optimise the overall IT infrastructure. One might also think that the connection missing in the Figure 3 is the one between the VCs. The problem would indeed be easier to tackle if VCs were able to exchange their workload and find agreements between themselves that meet their own preferences.

Acknowledgement: This work was supported, in part, by Science Foundation Ireland grants 10/CE/11855 and 13/RC/2094. Anthony Ventresque was supported, in part, by Science Foundation Ireland Industry Fellowship grant 13/IF/12789. Ivona Brandic was supported, in part, by the TU Vienna Distinguished Young Scientist Award 2011.

REFERENCES

- [1] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *FGCS*, pp. 755–768, 2012.
- [2] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, pp. 11–25, 2015.
- [3] A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: A real case based on openstack cloud," *FGCS*, pp. 118–127, 2014.
- [4] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *CloudCom*, pp. 91–98, 2011.
- [5] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *International Symposium on Integrated Network Management*, pp. 119–128, 2007.
- [6] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *ICS*, pp. 175–184, 2008.
- [7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamism of clouds at scale: Google trace analysis," in *SoCC*, p. 7, 2012.
- [8] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya, "Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud data-center," in *ICA3PP*, pp. 371–384, 2011.
- [9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Networked Systems Design & Implementation (NSDI)*, pp. 273–286, 2005.
- [10] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI*, pp. 17–17, 2007.
- [11] F. Lopez Pires and B. Baran, "Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach," in *UCC*, pp. 203–210, 2013.
- [12] A. Sallam and K. Li, "A multi-objective virtual machine migration policy in cloud systems," *The Computer Journal*, pp. 195–204, 2014.
- [13] G. Jung, M. Hiltunen, K. R. Joshi, R. D. Schlichting, C. Pu, et al., "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *International Conference on Distributed Computing Systems (ICDCS)*, pp. 62–73, 2010.
- [14] J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," in *ICAC*, pp. 225–234, ACM, 2011.
- [15] H. Zhao, Z. Yu, S. Tiwari, X. Mao, K. Lee, D. Wolinsky, X. Li, and R. Figueiredo, "Cloudbay: Enabling an online resource market place for open clouds," in *UCC*, pp. 135–142, 2012.
- [16] Y. Kessaci, N. Melab, and E.-G. Talbi, "A Pareto-based genetic algorithm for optimized assignment of virtual machines requests in a multi-cloud environment," in *CEC*, 2013.
- [17] "Google/roadef challenge '12." <http://challenge.roadef.org/2012/en/>.
- [18] T. Saber, A. Ventresque, J. Marques-Silva, J. Thorburn, and L. Murphy, "Milp for the multi-objective vm reassignment problem," in *ICTAI*, 2015.
- [19] X. Li, A. Ventresque, J. Omana Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *HPCS*, 2015.
- [20] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *CloudCom*, pp. 254–265, 2009.
- [21] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *GreenCom*, pp. 179–188, 2010.
- [22] C.-H. Lien, Y.-W. Bai, and M.-B. Lin, "Estimation by software for the power consumption of streaming-media servers," *Transactions on Instrumentation and Measurement*, pp. 1859–1870, 2007.
- [23] T. Saber, A. Ventresque, X. Gandibleux, and L. Murphy, "Genepi: A multi-objective machine reassignment algorithm for data centres," in *HM*, pp. 115–129, 2014.
- [24] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, pp. 109–133, 1995.
- [25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Transactions on Evolutionary Computation*, pp. 182–197, 2002.
- [26] A. Alsheddy and E. E. Tsang, "Guided pareto local search based frameworks for biobjective optimization," in *CEC*, pp. 1–8, 2010.
- [27] E. Zitzler, M. Laumanns, L. Thiele, C. M. Fonseca, and V. G. da Fonseca, "Why quality assessment of multiobjective optimizers is difficult," in *GECCO*, pp. 666–673, 2002.
- [28] E. Zitzler, J. Knowles, and L. Thiele, "Quality assessment of pareto set approximations," in *Multiobjective Optimization*, pp. 373–404, 2008.
- [29] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms — a comparative case study," in *PPSN*.
- [30] K. Bringmann and T. Friedrich, "Approximation quality of the hypervolume indicator," *Artificial Intelligence*, vol. 195, pp. 265–290, 2013.