GeNePi: a Multi-Objective Machine Reassignment Algorithm for Data Centres

Takfarinas Saber[1], Anthony Ventresque[1], Xavier Gandibleux[2] and Liam Murphy[1]

[1] Lero@UCD, School of Computer Science and Informatics, University College Dublin, Ireland
takfarinas.saber@ucdconnect.ie, {anthony.ventresque, liam.murphy}@ucd.ie
[2] Faculty of Science, University of Nantes, France
xavier.gandibleux@univ-nantes.fr

Abstract. Data centres are facilities with large amount of machines (i.e., servers) and hosted processes (e.g., virtual machines). Managers of data centres (e.g., operators, capital allocators, CRM) constantly try to optimise them, reassigning 'better' machines to processes. These managers usually see better/good placements as a combination of distinct objectives, hence why in this paper we define the data centre optimisation problem as a *multi-objective machine reassignment problem*. While classical solutions to address this either do not find many solutions (e.g., GRASP), do not cover well the search space (e.g., PLS), or even cannot operate properly (e.g., NSGA-II lacks a good initial population), we propose *GeNePi*, a novel hybrid algorithm. We show that GeNePi outperforms all the other algorithms in terms of quantity of solutions (nearly 6 times more solutions on average than the second best algorithm) and quality (hypervolume of the Pareto frontier is 106% better on average).

Keywords: Data Centres, Machine Reassignment, Evolutionary Algorithms, Multi-Objective Optimisation.

1 Introduction

Data centres are facilities dedicated to hosting many computer resources, and while they have been around for decades, they are now the centre of attention as they are increasingly the crucial element of our digital lives (e.g., for the Cloud). These data centres evolve constantly as for instance machine age and are eventually decommissioned, new ones (more powerful) are bought regularly, and processes hosted are updated to potentially more greedy ones. Managers of data centres have to adapt their systems to these evolutions and migrate processes from one machine to another according to technical and non-technical reasons, what we call *reassignment of processes to machines*. For instance, managers may want to increase the reliability of their data centre and move workload from overloaded machines to less loaded and/or more powerful ones. Often, they also

try to move services to power efficient machines, in order to lower the cost and environmental impact of the data centres.

One problem is that machines can range to up to tens of thousands (e.g., OVH, a European leader in the domain, have 150,000 servers in 12 data centres[3]), and services up to millions (e.g., VMware ESX accepts up to 320 VMs per host). At this scale, any instance of the reassignment problem becomes a challenge to the existing heuristics and solvers, and finding the 'best' (re-)assignment an illusion. Another problem is that, as we mentioned in the previous paragraph, managers have different perspectives on what is a 'good' solution, and ranking all the solutions according to a single utility function (e.g., minimising energy consumption) is probably not relevant.

This is a perfect example of a problem where *multi-objective decision making* makes sense: an optimisation problem with various independent objectives that only decision makers can compare – possibly collectively. For instance, Xi et al. [1] describe such an enterprise environment where managers of virtual data centres have various perspectives when it comes to placement decisions. Hence we call the problem we address in this paper *multi-objective optimisation for the machine reassignment problem*. While this problem has been addressed in the context of machine assignment [2], or for dynamic assignment of small amount of machines [3], it has not been in itself the topic of research in the past. In this paper we identify three objectives for the problem: (i) reliability, i.e., a penalty is given to assignments that load too much the machines; (ii) migration, i.e., assignments that move processes too much (especially to remote locations) are penalised; and (iii) electricity: trying to obtain assignments that minimise the (electrical) cost of running the data centre.

In this paper we show that the classical solutions do not perform well against this problem, in terms of the number of non-dominated solutions found (the *quantity* of solutions) or the hypervolume [4] of the search space area defined by the Pareto frontier (the *quality* of the solutions). Pareto Local Search (PLS) [5,6,7] usually finds solutions but they are grouped in one area of the search space (small hypervolume) and it is a slow algorithm – these are the expected behaviour of this algorithm. NSGA-II [8] needs a good initial population in order to operate properly, while here it gets only one solution: the initial assignment. GRASP [9] does not perform well in such large search spaces and ends up trying a lot of non-feasible settings, eventually finding few or no solutions. We then propose a novel hybrid algorithm called *GeNePi*, using successfully three steps: a first step (inspired from GRASP) to explore quickly all the search space, a second (using NSGA-II) to introduce some variety and quality in the solutions and a last one (PLS-based) to increase the number of solutions. GeNePi outperforms all the state-of-the-art other algorithms (the previously mentioned ones and some classical bin packing ones), finding nearly 6 times more non-dominated solutions on average and covering the search space better (106% better on average).

In the rest of this paper we first give a problem definition, with the constraints and the three objectives that we identified as the most relevant (Section 2).

[3] Source: http://www.ovh.com/fr/backstage/ – accessed on 23/11/2013.

Then we describe GeNePi, our algorithm for solving this multi-objective machine reassignment problem (Section 3). After this, section 4) proposes an evaluation of GeNePi against the state-of-the-art other algorithms. Finally we make some concluding remarks (Section 5).

2 Problem Definition

The *Multi-Objective Machine Reassignment Problem* consists in optimising the usage of a set of machine $\mathcal{M}$ according to various objectives. More specifically, reassignment in general seeks to find a new machine $M(p)$ for every process p in the system, initially placed in machine $M_0(p)$, satisfying the constraints of the system while its multi-objective version tries to find non-dominated (better than every other solution in some directions of the space). In some cases $M_0(p) = M(p)$, which means that the process p does not move during the reassignment.

The machine reassignment problem can be seen as a complex (i.e., with more constraints such as dissemination and dependency) instance of *d-dimensional vector bin packing* [10,11] with each machine a *d-dimensional bin*, such that d is the number of the resources. The aim is to place the processes in these machines, such a way they satisfy their capacities and they minimise the number of bins used. This problem is $NP - Hard$ and it has drawn lot of attention [12]; in this work we do not consider the scheduling aspects of it (*bin repacking scheduling problem* [13]). The model we describe below is loosely inspired by several work (e.g., [14] for a linear model), among which the problem definition of the ROADEF challenge [15] has an important place.

2.1 Reassignment Problem

A machine $m \in \mathcal{M}$ belongs to a location $l \in \mathcal{L}$ (the site where the server is located). It is also in a neighbourhood $N(m) \subseteq \mathcal{M}$, which represents a set of machines with which it is linked to by fast connections or with which it shares the same protocol. Each machine belongs to one and only one location and one neighbourhood. Every m has also several resources $r \in \mathcal{R}$ (e.g., RAM, CPU, disk), in limited capacities $Q_{m,r}$. We consider that the quantity of resource r that the process p needs is fixed to $d_{p,r}$ and corresponds to a VM parameter/SLA[4]. The first constraint regards the number of processes a machine can host:

$$\sum_{p \in \mathcal{P} \mid M(p)=m} d_{p,r} \quad \leq \quad Q_{m,r}, \quad \forall m \in \mathcal{M}, \forall r \in \mathcal{R} \tag{1}$$

Some resources are called *transient*: $r \in \mathcal{TR} \subseteq mathcalR$. Such resources (e.g., RAM and disk) are needed on both machines during a new assignment, as

[4] a Service-Level Agreement (SLA) is a contract agreed between a data centre provider and a customer which describes the service provided (e.g., allocated resources, time to recover after an outage).

the processes use the resources on both machines during the migration.

$$\sum_{p \in \mathcal{P} | M_0(p)=m \vee M(p)=m} d_{p,r} \ \leq Q_{m,r}, \quad \forall m \in \mathcal{M}, \forall r \in \mathcal{TR} \tag{2}$$

Other resources are called *non-transient*: $r \in \mathcal{NR} \ = \ \mathcal{R} \setminus \mathcal{TR}$.

Services/applications are often multi-tier (e.g., to separate concerns) and replicated (for performance and security reasons), so it is realistic to assume here that processes (the atomic element of workload) are organised by services. It is a common for services to have an *anti-cohabitation* constraint [16], i.e., the processes composing a service cannot share the same host – for some reliability, security and performance reasons. Let $\mathcal{P}$ be the set of processes and $\mathcal{S}$ the set of services, then the *anti-cohabitation* constraint can be expressed as in (3).

$$\forall p_i, p_j \in \mathcal{P}, i \neq j, \forall s \in \mathcal{S}, (p_i, p_j) \in s^2 \Rightarrow M(p_i) \neq M(p_j) \tag{3}$$

For the same reasons of reliability, security and performance, services require that the number of locations hosting at least one process has to be greater than a certain number, caller *spread number*. This allows increasing the resilience in case of failure of a data centre: the bigger the spread number, the safer the service.

$$\sum_{l \in \mathcal{L}} \min\left(1, |\{p \mid p \in s \ \wedge \ M(p) \in l\}|\right) \geq spreadNumber_s, \quad \forall \ s \in \mathcal{S} \tag{4}$$

Services can also depend on each other and in this case the processes of these services need to be close to each other – to increase the performance of the system. We note this dependency of services $\hookrightarrow$. Of course, as the dependencies between services can be complex, the assignment can be tricky: a process $p \in \mathcal{P}$, belonging to service $s_i \in \mathcal{S}$ which is dependent on service $s_j \in \mathcal{S}$ and service $s_k \in \mathcal{S}$, needs to be assigned to a machine in $N(m)$ with $\exists p' \in s_j \cap s_k m = M(p')$.

$$\forall s_i, s_j \in \mathcal{S}, \ s_i \hookrightarrow s_j \implies \forall p_u \in s_i, \ \exists p_v \in s_j \mid N(M(p_u)) = N(M(p_v)) \tag{5}$$

Figure 1 shows graphically a scenario (i.e., instance and initial solution) of the problem. Note that resource capacities and demands are not represented here to make it simpler to understand.

Definition 1 (Machine Reassignment). *An assignment A of processes to machines is a mapping: $A : \mathcal{P} \mapsto \mathcal{M}$, such that $A(p, \mathcal{M}) \rightarrow m$, which satisfies all the previous constraints 1, 2, 3, 4 and 5.*
A reassignment is a function that modifies an initial assignment: $ReA : A \mapsto A$ and gives a new assignment of processes to machines.

2.2 Objectives

As said in the introduction, there are several perspectives on the best optimisation, which translate in our case into several objectives. Some studies [17]

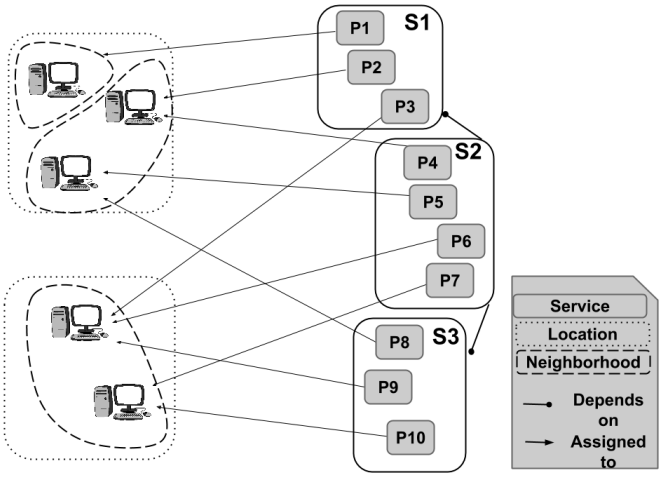

Fig. 1. Simple scenario of a correct assignment of processes to machines (spread= 2).

show that a large number of objectives decreases drastically the performance of evolutionary algorithms, and that decision makers tend to favour small number of dimensions. We have then decided to focus only three objectives that seem to make the most sense from the literature and discussions with industrials: reliability, migration and electricity costs.

There are many elements that can help data centre operators to predict the risk of failure of a server: to name a few the age of a machine, the vendors of its parts (e.g., processor maker) and the past history of similar machines. They are complex to collect and understand, and we do not know exactly how to process them to obtain an objective that the data centre operators and decision makers could use (the literature seems uncertain on the matter [18]). One thing we know is that as opposed to the risk of failure, the *reliability* is easier to compute and gathers less questions. Machines do operate better when they are not too loaded, and reliability can be estimated through the load: the more loaded a machine, the greater the risk of performance issues or failures.

Definition 2 (Reliability Cost). *A machine $m \in \mathcal{M}$ is reliable if it is not loaded more than a reliability value $\rho(m, r)$ for each resource $r \in \mathcal{R}$, and we compute a reliability cost of m, $\rho(m)$, as:*

$$\rho(m) = \sum_{r \in \mathcal{R}} \max \left(0 \quad , \sum_{p \in \mathcal{P} | M_0(p) = m \vee M(p) = m} d_{p,r} \; - \; \rho(m, r) \right) \tag{6}$$

If the *safety capacity* of m for the resource r is higher than the sum of the demands, then it does not impact the safety of the machine. Note that this definition is inspired by the concept of *safety capacity* introduced in [19]: if one

or several resources of a machine are over-loaded then the machine may not be able to satisfy its SLAs.

Migrating a process has a cost which is often neglected by research in the area but is well known by practitioners [20]. Basically, this consists in the time needed to prepare a process p for a migration ($\mu_1(p, M_o(p))$), to transfer p ($\mu_2(p, M_o(p), M(p))$) and to install p on a new machine ($\mu_3(p, M(p))$). All these costs are dependent on some process parameters (e.g., size of the data stored on disk and RAM, complexity of the installation) and topology parameters (e.g., number of hops, bandwidth), that we do not evaluate in this paper.

Definition 3 (Migration Cost). *The cost of migrating a process $p \in \mathcal{P}$ from a machine $M_0(p)$ to a machine $M(p)$ is defined:*

$$\mu(p, M_o(p), M(p)) = \mu_1(p, M_o(p)) + \mu_2(p, M_o(p), M(p)) + \mu_3(p, M(p)) \quad (7)$$

Electricity cost of running machines accounts for up to 50% of their operating costs [21] and it is a burden for countries' electricity production systems: in 2007, Western European data centres consumed 56 TWh of electricity, and this is expected to double (104 TWh, or about 4 times the annual production or Ireland) by 2020 [22]. There is a global trend towards more greener and power-aware practices, and this will certainly lead to an increase in the electricity price and other incentive for data centre managers to minimise their electricity consumption. Modelling electricity cost is complex but we follow the general assumption that states that it is a linear function of its CPU usage [23,24]. We then just define two parameters, α_m (linear factor) and β_m (fixed cost of running m with n load on the CPU) for every machine m. This does not take into account other elements that may be relevant but are somehow out of the scope of our study here (e.g., cooling of data centres).

Definition 4 (Electricity Cost). *The electricity cost of a machine $m \in \mathcal{M}$ in the location $l \in \mathcal{L}$ depends on the variables α_m, β_m (electricity consumption constants) and γ_l (electricity cost in l), and is expressed by the following formula:*

$$\epsilon(m) = \begin{cases} \gamma_l \times \left(\alpha_m \times \sum_{p \in \mathcal{P} | M(p) = m} d_{p,CPU} \; + \; \beta_m \right) & \text{if } m \text{ is running} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

3 Description of our Solution: GeNePi

GeNePi applies successively three (modified) optimisation algorithms: GRASP, NSGA-II and PLS. This idea of using three steps for approximate resolution [25] is new in the domain of data centres optimisation.

3.1 Ge: a variant of the constructive phase of GRASP

We use a variant of the constructive phase of *Greedy Randomized Adaptive Search Procedure* [9] (GRASP). Solutions are generated by trying to reassign processes

one after the other, according to a greedy heuristic which is slightly relaxed to include a random factor. This method is commonly used for combinatorial problems, and applied to get some quick initial solutions with good objectives. After ranking the processes according to their dependencies and their needs of resources, they are selected one by one. A decision of reassigning one per cent of the processes from their initial hosts has been taken, because of the tightness of transient resource constraints that limits the number of reassignments. The choice of the reassignment of every process is based on a linear combination of the three utility/objective functions (one per objective). Even if a linear combination of these utility functions allows us to go beyond the objective types barrier, its static definition induces getting solutions with a same objectives level of interest. This behaviour goes against the aim of a multi-objective optimisation. That is why we adopted a panel of triplet weights (λ_i, λ_j, λ_k) in $]0,1]^3$, with $\lambda_k = 1 - \lambda_i - \lambda_j$. They are chosen in such a way they cover a maximum search space by optimising the objectives separately in addition to their trade-offs. They will be used to introduce a diversification in the interest of each objective, ensuring a trade-off between them. The random part of GRASP lays in the assignment of a machine to each process, at each iteration. For each process, a set of assignable machines that respect the constraints is computed, and a value of interest is given to each machine by a weighted sum: (U_i): $\sum_{i=1}^{3} \lambda_i U_i$,, which creates a set of machine with a utility lower than or equal to $(minUtility + (1-r) * [maxUtility - minUtility])$, with $r \in [0,1]$. A random machine is selected from this eligible set to assign the process to it. During the assignment, it may happen that a process has no machine able to host it. The solution is declared infeasible, and removed from the initial solutions. Globally, at the end of this step, we expect to have a set of decent solutions spread over the search space.

3.2 Ne: NSGA-II

We use for this step a genetic algorithm called *Non-dominated Sorting Genetic Algorithm-II* [8] (NSGA-II). This step is useful for the improvement of the *Pareto set*[5] obtained from the first step. This metaheuristic allows to get a good dissemination of the solutions around the Pareto frontier and prevent their accumulation in some area of the search space. Hopefully, it allows GeNePi getting a smooth frontier and increases the number and the quality of the non-dominated solutions. It is a genetic algorithm, i.e., it runs an evolutionary process which matches individuals (i.e., solutions or assignments) at each generation and mixes their features (as the biological evolution would do with genes). The two main actions are *crossover* which mixes genes from two parents, and *mutation* that creates randomly individuals with new features. There exists several ways of doing crossovers, which is more or less a cut and paste operation where assignments in the set of actual solutions are split into regular length segments and swapped

[5] Pareto set: a set of non-dominated solutions (i.e., better than all other solutions in one or more objectives).

with one another [26]. In our case crossovers consider the exchange of services (group of processes) rather than blocks of process assignments – that minimise the number of bad crossovers. Of course the diversity is less than with crossovers on processes, but we compensate with a bigger probability of mutations (i.e., random assignments in solutions to see whether this improve the utilities). After a generation has "passed", some new individuals are kept (usually the fittest, those with the best objective values: low domination rank, but also some other that allow to introduce some variety: high crowding-measure [8]), and others are suppressed. Hence the global population of assignments only improves (descendants worse than their parents are likely to be suppressed). Beside, last generations tend to be well distributed over the Pareto frontier.

3.3 Pi: a Pareto Local Search

Finally, we try to improve the Pareto set by using a *Pareto Local Search* [5,6,7] (PLS). It consists in applying several local search operators on the solutions belonging to the *Pareto* frontier. Few simple moves are chosen to analyse the neighbourhood of actual solutions: (i) swap, i.e., taking two processes and exchanging their assignment; (ii) 1-exchange, where one process at a time is selected and reassigned to any machine that accepts it; (iii) shift, where processes belonging to the same service exchange their assignments (which maintains the satisfaction on the dependency constraints). These moves allow probing of a large neighbourhood around the current solutions, which may generate some redundancies if the solutions are close of one another. To overcome this problem, we generate boxes by clustering solutions, and apply a local search to the most isolated solution in each of them (i.e., has the largest crowding-measure value). Only one neighbourhood is generated for every selected solution at every iteration, even if new interesting solutions have been found. This balances the improvement and reduces the execution time as redundancy is less likely.

4 Evaluation

In this section, we evaluate the performance of our solution against other state-of-the-art multi-objective and reassignment solutions, using several metrics: time, quantity (number of solutions) and quality of solutions (hypervolume). We create a benchmark inspired by the ROADEF Challenge 2012 [19].

4.1 Experimental Setups

The ROADEF challenge 2012 is particularly suited to our needs, as it is rather realistic (proposed by Google, who claim it represents accurately some of their data centres) and it is quite comprehensive: lot of resources for the machines/processes while many papers in the area only consider two (namely, RAM and CPU), reasonably high number of machines and processes, complex dependencies and constraints on the services and processes which make the assignments not straight-

forward. The ROADEF dataset distinguishes three categories of instances (a_1 are considered 'easy', a_2 'medium' and b 'hard').

In this paper, we pick up 14 instances (see Figure 1), leaving only the biggest ones. We have added variables α_m and β_m to each machine $m \in \mathcal{M}$, and γ_l for every location $l \in \mathcal{L}$ in order to include electricity consumption. All tests were made on a 4 cores Intel Xeon 3.10GHz CPU, with 8GB of RAM, running Ubuntu 12.4 LTS 64-bits.

Instance	# Resources	# Machines	# Services	# Processes	Execution Time (s)
a_1_1	2	4	79	100	1.58
a_1_2	4	100	980	1,000	3,108
a_1_3	3	100	216	1,000	441
a_1_4	3	50	142	1,000	309
a_1_5	4	12	981	1,000	332
a_2_1	3	100	1,000	1,000	3,905
a_2_2	12	100	170	1,000	600
a_2_3	12	100	129	1,000	695
a_2_4	12	50	180	1,000	342
a_2_5	12	50	153	1,000	347
b_1	12	100	2,512	5,000	14,990
b_2	12	100	2,462	5,000	10,028
b_3	6	100	15,025	20,000	39,595
b_4	6	500	1,732	20,000	63,534

Table 1. The dataset used for our evaluation (ID and size of the different instances) and execution time of GeNePi on them.

4.2 Metrics

Comparing multi-objective optimisation approaches is complex as the set of solutions they give on a problem can be seen from different perspectives: coverage, closeness to the Pareto frontier, variety, and many more [27]. The problem probably roots in the fact that the Pareto frontier is unknown most of the time, and that the different objectives cannot be taken in isolation to give the quality of any solution. In this paper, we made the decision to take only few unary operators as metrics (see other studies for a more comprehensive study of the various possible operators [28]): unary as they take a set of solutions and give a single value, which allows comparing the different approaches.

The first metric we use is the number of *non-dominated (efficient) solutions* and we refer to it as the quantity of solutions. Finding a large number of solutions is always better as it provides more alternatives to the decision makers.

The other metric is the *hypervolume* [4] (also known as the $\mathcal{S}$ metric). We sometimes call it quality of the solutions. This is a widely used metric in the area of optimisation to evaluate the performance of multi-objective algorithms

that aims at understanding how the output sets are spread in the different dimensions. In short, the hypervolume measures the space (in the n dimensions of the n objectives) defined by the set of non-dominated solutions and a reference point, picked in the space as far as possible from the Pareto frontier. The bigger the hypervolume, the more interesting are the solutions in the found non-dominated solution set, as they increase the dominated area. In formal terms, this is proven by Fleischer [29] who states that the maximisation of the hypervolume is equivalent to finding the optimal Pareto frontier. Note that in order to compare the result sets of different algorithms, we use the same reference points for each instance of the multi-objective machine reassignment problem. A last remark is that although hypervolume is obviously impacted by the random seeds applied to algorithms, a preliminary study we conducted did not give us any sense that the relative results of the different algorithms would vary. In other words, good algorithms are always good, bad are always bad, whatever the seed - in particular GeNePi always gives the best hypervolume values. We hence use the same seed for each algorithms, but plan to study in more details the impact of each technique on the Pareto front in some future work.

4.3 Algorithms

We compare our solution to three different types of algorithms, running for the same period of time. The first algorithms are from the *First Fit family*. These heuristics are designed for Vector Bin Packing [30] and they are considered efficient. We chose among them the First Fit (FF) which selects the first machine that fits for every process in a sequence; the Random Fit (RF) which selects randomly a machine among those which fit; and the First Fit Descent Bin-Balancing (BB) which selects the least loaded machine for each process.

The second set of algorithms is the state-of-the-art solutions from the multi-objective optimisation field. The first of them is GRASP in its original definition, i.e., the choice of reassigning is based on a uniform probability distribution on the eligible machines. We also run the first step of GeNePi (Ge) as it is a variation on GRASP that we expect is better than GRASP for our scenario. We also try NSGA-II where we reserve a third of the execution time to GRASP in order to create an initial population and run NSGA-II in the two remaining thirds of the execution time. The last Algorithm is a Pareto Local Search (PLS), with a number of boxes at every iteration equal to the number of solutions in the non-dominated solution set.

4.4 Tuning the steps of GeNePi

Each of the three steps composing GeNePi has several parameters that need to be tuned, and globally we need to decide how many iterations or how much time we allocate to each of them to make the best use of each. The tuning has been performed on the instance a_1_5.

The first step of GeNePi is Ge (based on GRASP), which has only one value to tune: α, the factor leading to more randomised greedy search (bigger α) or

local search (smaller α). We conducted a thorough evaluation of the impact of different values of α from 0.05 to 0.95 (repeated 10 times). The best value of α seems to be 0.6 regardless to the number of iterations. For Ne (i.e., NSGA-II), we combined 9 possible values $\{0.1, 0.2, \ldots, 0.9\}$ for P_c and P_m, obtaining 81 different variations of the parameters (we again run 10 times each combination). We realise that P_c values between 0.6 and 0.8 give better results, while the impact of P_m seems less important (values of P_m between 0.1 and 0.3 giving slightly better results though). We then decided to use $P_c = 0.6$ and $P_m = 0.2$. Pi has only one parameter that we can tune here: the number of zones (boxes) that it can explore. This number of zones has an impact on the quality of the Pareto frontier, and hence on the hypervolume. A small number of zones means less neighbourhood probing, but also less redundancy and execution time, while more zones allow to analyse more neighbourhoods (and to find more solutions) but there is a cost in redundancy and execution time. 10 seems to us a good trade-off between probing a large search area and reducing the execution time. Table 2 summaries the tuned parameters for each part of GeNePi.

GeNePi aims at providing decision makers with an important set of good solutions, covering the solutions space, and in a reasonable time. These two ideas (quality and time) seem to be incompatible, but they just force to consider time in a different way. In particular, Ne/NSGA-II needs to have a set of good initial solutions, and then we have to make sure Ge/GRASP has enough time. It appeared from tests that served to define α that a number of iterations from 100 to 500 lead to practically the same hypervolume. This is why; we picked 100 as the number of iteration for Ge. The good number of iterations for Ne/NSGA-II is much trickier to find as it depends greatly on the quality of the initial population. It seems, experimentally, that 100 iterations with a population size of 50 give good results, so this is what we use for Ne/NSGA-II. Pi/PLS is the most time consuming part, we use it only for one iteration in order to get a smooth Pareto.

Ge (1^{st} step - GRASP)		**Ne (2^{nd} step - NSGA-II)**		**Pi (3^{rd} step - PLS)**	
α	0.6	Probability of crossover	0.7	# zones (# boxes)	10
$\lvert\Lambda\rvert$	4	Probability of mutation	0.3	# iterations	1
# iterations	100	Size of population	50		
		# iterations	100		

Table 2. Parameters for the different steps of GeNePi after a tuning study.

4.5 Results

Table 3 shows that GeNePi outperforms notably all the other algorithms, both in terms of number of solutions (5.84 times more solutions found than the second best on average) and hypervolume (106% better on average). We notice that solutions from the first fit family have acceptable results only for some instances

	RF	FF	BB	GRASP	Ge	NSGA	PLS	GeNePi
a_1_1	4(2.6)	10(2.95)	87(3.73)	20(2.73)	42(3.6)	14(2.8)	10(2.4)	**193(3.95)**
a_1_2	1(7.49)	1(7.49)	1(7.49)	1(7.49)	26(8.47)	1(7.49)	2(7.49)	**133(9.14)**
a_1_3	1(4.17)	1(4.17)	1(4.17)	1(4.17)	19(4.27)	1(4.17)	2(4.17)	**62(4.32)**
a_1_4	1(9.72)	1(9.72)	1(9.72)	1(9.72)	40(11.1)	1(9.72)	2(9.72)	**187(12.2)**
a_1_5	4(2.51)	10(2.51)	2(2.45)	14(2.59)	49(2.74)	16(2.57)	32(2.52)	**66(2.78)**
a_2_1	33(4.86)	41(4.95)	1(4.57)	69(5.41)	57(5.43)	71(5.46)	4(4.61)	**227(5.68)**
a_2_2	1(1.33)	1(1.33)	1(1.33)	1(1.33)	22(1.55)	1(1.33)	2(1.33)	**171(1.76)**
a_2_3	1(2.02)	1(2.02)	1(2.02)	1(2.02)	30(2.36)	1(2.02)	67(2.04)	**250(2.67)**
a_2_4	1(6.42)	1(6.42)	1(6.42)	1(6.42)	28(7.62)	1(6.42)	2(6.42)	**374(8.63)**
a_2_5	1(9.91)	1(9.91)	1(9.91)	1(9.91)	28(10.3)	1(9.91)	2(9.91)	**245(11)**
b_1	1(8.2)	1(8.2)	1(8.2)	1(8.2)	27(8.34)	1(8.2)	39(8.34)	**207(8.49)**
b_2	1(1.43)	1(1.43)	1(1.43)	1(1.43)	23(1.48)	1(1.43)	2(1.43)	**300(1.53)**
b_3	1(6.25)	1(6.25)	1(6.25)	1(6.25)	20(6.27)	1(6.25)	108(6.27)	**162(6.3)**
b_4	1(3.65)	1(3.65)	1(3.65)	1(3.65)	22(3.67)	1(3.65)	3(3.67)	**118(3.7)**

Table 3. 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.

(a_1_1, a_1_5 and a_2_1). In general those algorithms favour the reassignment of the processes, with the side effect that more transient resources are consumed and more anti-cohabitation and dependency constraints are violated. That is why they perform better with instances that have larger transient resource capacities and a ratio between the number of processes and services close to one – this corresponds exactly to the three instances mentioned above. The same behaviour is observed for GRASP, which tends to reassign processes instead of keeping them on their initial assignment, as a decision is made based on a basic draw among several relevant machines for every process, based on a utility function. Hence the probability of choosing the initial assignment is low and GRASP tries a lot of solutions that end up being infeasible. NSGA-II, being dependent on the quality of the initial population, performs badly – although we give a partial result of GRASP to help it at the start. This is a major (but well known) drawback for this algorithm, especially for our scenario for which NSGA-II is clearly not fitted. The results for PLS are contrasting as they can be good in terms of quantity (better than Ge at times) but are poor in terms of quality – hypervolume values for PLS are always among the lowest. This comes from the fact that PLS searches for possible solutions locally, and may find some, but they are similar to the original ones and do not increase the diversity of the solutions set. For a multi-objective problem like ours, PLS is then not fitted either. Ge, the first step of GeNePi gets a good hypervolume but not an outstanding number of solutions. This was expected as it is only an improvement of GRASP which itself suffers from a lack of solutions. GeNePi is by far the best algorithm, and we explain it by the composition of elements: Ge (i.e., modified GRASP) finds a large number of solutions, allowing NSGA-II (the second step to operate properly

and finding new solutions that compromise all the objectives, while PLS, the last step, increase the number of solutions around the previously found ones.

Table 1 shows that GeNePi works in a short time for the easy and medium instances, and in a reasonable time (for our scenario) for the bigger ones. The 17 hours of running GeNePi for the biggest instance we consider (b_4) are totally justified if this can save money, increase the reliability and do not put the data centre at risk by performing too many migrations. Especially as GeNePi can give 118 solutions for this instance, i.e., 118 options for the operators to make the most informed decision. To give the reader a sense of what happens during GeNePi and specially the impact of the three phases, we plot the improvement curve of the instance a_1_5 (see Figure 2). Each point corresponds to one or several new non-dominated solutions found (with the timestamp of this new solution in x-axis and the new hypervolume of the solution set in y-axis). We can see that GRASP finds solutions quickly (9 s.) and the rise in the hypervolume signifies that they are distributed in the search space (which is good). GRASP continues after the first 9 s. and finds new solutions, but we notice a quasi-stagnation of the hypervolume after 46 s. NSGA-II finds group of solutions, each group improving significantly the hypervolume: they correspond to a new search areas in the space discovered by the first solution of the group and improved by the others. PLS takes more time than the other steps, but it finds some new solutions and bring a little improvement in terms of hypervolume.

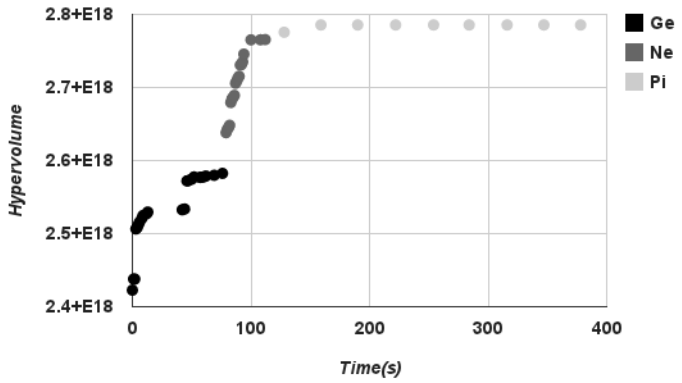

Fig. 2. Improvement curve of GeNePi on the instance a_1_5. Each point is a new solution (or a set of new solutions).

5 Conclusion

Reassigning processes to servers automatically is complex (lot of dimensions and constraints), large scale for most of the real instances (data centres are usually big

computing facilities) and needs to consider different objectives. In this paper we define the multi-objective machine reassignment problem and propose an hybrid solution using successively three optimisation steps: GeNePi. Multi-objective approaches are good when the set of possible solutions is large and extracting the 'best solution' is difficult. In this case, the system needs to be assisted by decision makers who can evaluate the different solutions with respect to their value in the different dimensions of the problem. Here, we defined the machine reassignment in the three dimensional space defined by: (i) reliability of the assignment, (ii) migration cost of the assignment, and (iii) energy consumption. Our solution, GeNePi, is based on three optimisations algorithms: Ge, a variant of the constructive phase of GRASP, which aims at finding an initial population with solutions representing every objective; Ne, based on a genetic algorithm called NSGA-II that mixes solutions of the initial population and tries to find new solutions (and more diverse ones); and Pi a local search that looks for more solutions in the neighbourhood of those that GeNePi has already found. We showed in a large experimental validation that GeNePi outperforms other state-of-the-art solutions: it finds 5.84 times more good (non-dominated) solutions that are scattered over more of the search space (hypervolume is 106% better) – which is desirable as we want to offer decision makers a large variety of different solutions. There are two things that we would like to explore more in some future work: electricity consumption which will need to incorporate more parameters (such as cooling of data centres) and SLAs.

Acknowledgement: supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

1. X. Li, A. Ventresque, N. Stokes, J. Thorburn, and J. Murphy, "ivmp: an interactive vm placement algorithm for agile capital allocation," in *CLOUD*, pp. 950–951, 2013.
2. K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *CloudCom*, pp. 91–98, 2011.
3. J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," in *CAC*, pp. 225–234, 2011.
4. E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms — a comparative case study," in *PPSN*, pp. 292–301, 1998.
5. E. Angel, E. Bampis, and L. Gourves, "A dynasearch neighborhood for the bicriteria traveling salesman problem," in *Metaheuristics for Multiobjective Optimisation*, pp. 153–176, 2004.
6. M. Basseur, "Design of cooperative algorithms for multi-objective optimization: application to the flow-shop scheduling problem," *4OR*, pp. 255–258, 2006.
7. A. Alsheddy and E. E. Tsang, "Guided pareto local search based frameworks for biobjective optimization," in *CEC*, 2010.
8. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *TEVC*, pp. 182–197, 2002.

9. T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *JGO*, pp. 109–133, 1995.
10. M. Gabay and S. Zaourar, "A GRASP approach for the machine reassignment problem," in *EURO*, 2012.
11. N. Bansal, A. Caprara, and M. Sviridenko, "Improved approximation algorithms for multidimensional bin packing problems," in *FOCS*, pp. 697–708, 2006.
12. T. Batu, R. Rubinfeld, and P. White, "Fast approximate {PCPs} for multidimensional bin-packing problems," *Information and Computation*, pp. 42 – 56, 2005.
13. F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," in *CP*, pp. 27–41, 2011.
14. D. Mehta, B. O'Sullivan, and H. Simonis, "Comparing solution methods for the machine reassignment problem," in *CP*, pp. 782–797, 2011.
15. "Google/roadef/euro challenge 2012: Definition of the machine reassignment problem." http://challenge.roadef.org/2012/files/problem_definition_v1.pdf.
16. E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *ICDCS*, pp. 700–709, 2011.
17. R. C. Purshouse and P. J. Fleming, "On the evolutionary optimization of many conflicting objectives," *TEVC*, pp. 770–784, 2007.
18. B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *TDSC*, pp. 337–351, 2010.
19. "Google/roadef/euro challenge 2012." http://challenge.roadef.org/2012/en/.
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. D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan, "Comparing vm-placement algorithms for on-demand clouds," in *Dynamic data center power management: Trends, issues, and solutions*, 2008.
22. "Datacentre energy efficiency." http://re.jrc.ec.europa.eu/energyefficiency/html/standby_initiative.htm.
23. J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *GreenCom*, pp. 179–188, 2010.
24. C.-H. Lien, Y.-W. Bai, and M.-B. Lin, "Estimation by software for the power consumption of streaming-media servers," *TIM*, pp. 1859–1870, 2007.
25. X. Gandibleux, B. Martin, O. Perederieieva, and S. Rosembly, "Sur la résolution approchée en trois étapes du sac-à-dos bi-objectif unidimensionnel en variables binaires," in *ROADEF*, pp. 2–4, 2011.
26. E. Falkenauer, *Genetic algorithms and grouping problems*. 1998.
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, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *TEVC*, pp. 117–132, 2003.
29. M. Fleischer, "The measure of pareto optima applications to multi-objective metaheuristics," in *EMO*, pp. 519–533, 2003.
30. R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.