A genetic algorithm for cost-aware business processes execution in the cloud

Guillaume Rosinosky[1], Samir Youcef[1], and François Charoy[1]

Université de Lorraine, CNRS, Inria, LORIA F-54000 Nancy, France
guillaume.rosinosky,samir.youcef,francois.charoy@loria.fr

Abstract. With the generalization of the Cloud, software providers can distribute their software as a service without investing in large infrastructure. However, without an effective resource allocation method, their operation cost can grow quickly, hindering the profitability of the service. This is the case for BPM as a Service providers that want to handle hundreds of customers with a given quality of service. Since there are variations in the needed load and in the number of users of the service, the allocation and scheduling methods must be able to adjust the cloud resource quantity and size, and the distribution of customers on these resources. In this paper, we present a cost optimization model and an heuristic based on genetic algorithms to adjust resource allocation to the needs of a set of customers with varying BPM task throughput. Experimentations using realistic customer loads and cloud resources capacities show the gain of these methods compared to previous approaches. Results show that, in our case, using our algorithm on split groups of customers can provide better results.

Keywords: BPM, elasticity, cloud, optimization

1 Introduction

Consuming "Business Process Management as a Service" (BPMaaS) offer benefits that IT people widely acknowledge. It reduces the operational burden and allows to rely on the provider for the maintenance and provisioning of the service. However, from the BPMaaS provider point of view, it increases the operational complexity. The provider must ensure that all his customers receive the same attention and a defined quality of service at all time. He must also ensure that it operates at the best possible cost.

A Business Process Management System (BPMS) deployment is complex: it requires application servers, process engines and database management systems. Clustered installations requiring load balancers can also be deployed for high availability. Each customer has a different usage pattern and the number of tasks to execute evolves constantly. Each cloud compute resource is costly and has a capacity corresponding to its CPU, memory, storage and network speed, for a defined response time. In order to maintain an optimal infrastructure cost, processes executions must be distributed on different cloud resources. Figure

1 shows an example with two customers (or tenants). They require different capacity at each hour. We want to allocate the cheaper Cloud resources and distribute the process execution of tenants on these resources to optimize the operational cost.

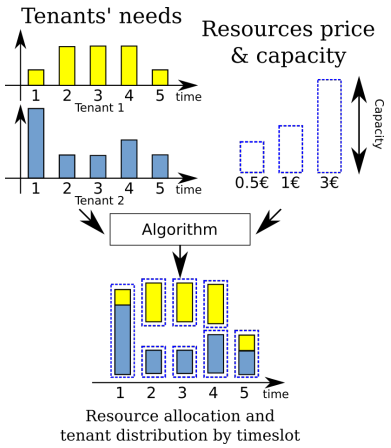

Fig. 1. Multi tenant resource allocation and distribution tenants

In order to optimize resource usage, we must sometimes migrate tenants from an installation to another. In figure 1, at time 5, tenant 1 is moved on the same resource as tenant 2 because both require less capacity and fit on this resource. A migration generates disruption of service on the customer side. We must stop the processes and move the data from one installation to the other [1]. We must find the best distribution of tenants on resources for each interval of time while controlling the number of migrations for each tenant. This problem becomes complex for a high numbers of tenants.

We propose an integer linear programming (ILP) model and a genetic algorithm that aims at finding the best allocation strategy while limiting the number of migrations per tenant. We improve our previous results [2] with a method that provides a better cost elasticity.

In the next section, we describe our migration strategy model, our genetic algorithm approach coupled to a solving of our model, and our previous iterative heuristic. We then study experimental results on both of the approaches compared to a baseline approach. We also show how we achieve interesting experimental results. Then, we compare our results with the state of the art. The last part concludes and presents our future work.

2 A migration strategy based model

In this Section we present the context for BPMaaS, our hypothesis and our method to optimize resource cost. This can be seen as a resource allocation problem with a constraint regarding the number of reallocations. Moving a tenant from a resource to another generates a service disruption. We want to limit their number to ensure stable quality of service for the customers. Thus, one of our problems is to find the right time to move tenants. We call it the migration strategy.

2.1 Context and constraints

Our approach is tenant-centric. All customers (tenants) processes are executed on the same BPMS installation. It is easier to manage deployments by customers rather than by processes. Customer processes share business data and security configuration that we need to manage together. Our assumptions are the following :

BPMS do not scale infinitely. There is no such thing as an infinitely scalable BPMS installation. Even clustered installation reach a limit due to the transactional nature of the database interactions. Thus, we use several BPMS installations. In our approach, we assume our tenants can fit on the "bigger" resource available.

Provisioning and deprovisioning takes time. We cannot change instantly tenant distribution, as computing instances instantiation, software installation, and data migration takes time. Thus we compute resource allocation in a discretized manner at fixed time interval or time slot. A time slot is a significant period of time for the provider: it could be a few seconds or an hour.

A tenant is a customer of the BPMaaS. Tenants run BPM processes composed of tasks. To execute them, the BPMS needs computing power, network bandwidth, disk and memory. It relies on separated compute instances for database systems for data persistence, and load balancers for clustered installations.

Task throughput is our performance metric. It corresponds to the number of BPM tasks executed for one period of time (e.g. per second). This metric is meaningful for the customers.

Our approach is offline. We assume we know the required BPM task throughput for each tenant and each time slot.

A cloud resource has a capacity expressed in BPM task throughput. A *cloud resource* (or *resource*) is one or several cloud compute instances that we use for the database tier, BPM system tier, load balancer tier, etc. It supports a full BPMS installation. A cloud resource can host several tenants. In our case, we assume a tenant fits on one resource: tenants won't be distributed on several cloud resources.

The number of migrations must be limited. We name *migration* the action of moving a tenant data and processes from one cloud resource to another. It requires the target cloud resource to be up and running. This action takes time to be executed. If all tenants of a cloud resource are migrated, it can be released. Migrations generate QoS breaks for the customers [1]. We limit their number for each tenant depending on the Service Level Agreement.

Without an optimization method, a solution to allocate tenant to resources would be to allocate each tenant on a resource that supports its maximum task throughput. We call this solution, *baseline method*. This method is often used in production, but can become very expensive.

We proposed in [2] a method based on an iterative heuristic and time series segmentation. It computes the list of necessary cloud resources and a mapping of tenants on each resource, time slot per time slot. We present it in the next Section.

2.2 Allocation with an iterative heuristic and time series segmentation

In this part, we recall briefly our previous method [2]. It has two parts: first, we choose a *migration strategy* i.e the time slots where each tenant can migrate, and, second, we apply a heuristic using resources prices and capacity, tenants needs, and their migration strategy in order to obtain the cloud resources and the placement of tenants.

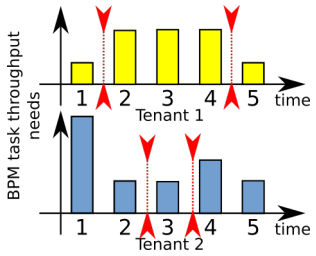

Fig. 2. Exemple of a migration strategy

For the first part, a time series segmentation [3] allows to select good migration times for each tenant. We present in figure 2 two examples of migration strategies, in red for two tenants. In this figure, *Tenant 1* can only be migrated at the end of time slots 1 and 4. For *Tenant 2*, it is time slots 2 and 3. This means both tenants will be migrated at most 2 times. They will stay on their origin resource if there is no migration authorization.

For the second part, our iterative timeslot heuristic is based on Variable Cost and Size bin packing in which we added repacking steps. It takes as input the tenant throughput by timeslot, the cloud resource prices and capacity, and the migration strategies for the tenants.

This coupled approach provides better results than the baseline approach. Still, the comparison with the solution computed with a solver showed that it is far from the optimal cost. It is possible to enhance this solution. We propose to use a genetic algorithm to find migration strategies that reduce the resource cost, and propose an alternative to the iterative timeslot heuristic based on integer linear programming. We present this model in the next Section.

2.3 An efficient model for migration strategies

Our problem is to find resource and tenant distributions for each time slot, for given tenants' loads, resource prices and capacity, and migration strategy. The model we propose answers to this problem. Our model principle is based on the absence of tenant migration when there no authorization to move. In this case, simple placement constraints should exist, and no constraint exist when there is an authorization to move in the migration strategy.

Let the following variables:

- $\mathcal{T}$, the set of cloud resource types, with t its cardinality.
- $\mathcal{I}$, the set of tenants with n its cardinality.
- $\mathcal{J}$, is $\mathcal{T} \times \mathcal{I}$ the set of all possible cloud resources associated with each tenant. its cardinality is $m = t \times n$.
- C_j, and W_j represent respectively the cost and the capacity in terms of BPM task throughput for the configuration j, with j in $\mathcal{J}$.
- w_i(k), the required capacity in terms of BPM task throughput for the tenant i during time slot k.
- $\mathcal{K}$ defines all the time slots, from 0 to D, where $D+1$ is the number of time slots.
- ${x_j}^i(k)$, the assignment of tenant i to configuration instance j during time slot k.
- $y_j(k)$, the activation of configuration j during time slot k.
- M, the maximum number of migrations of tenants between cloud resources on all time slots.
- $h_i(k)$ with $0 \leq k \leq D-1$. $h_i(k)$ is equal to 0 if the tenant i is not allowed to be migrated between time slot k and $k+1$, and equal to 1, if it is allowed. The set of all $h_i(k)$ (for each tenant and each time slot) is a migration strategy.
- migration strategies assume the maximum number of migrations allowed per tenant: $\forall i \in \mathcal{I} \sum_{k}^{k \in \mathcal{K}} h_i(k) = M$ where M is the number of migrations. This number depends on the SLA.

The objective is to minimize the total cost for all active cloud resources, for each time slot, as shows equation 1.

$$\min \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K}} C_j y_j(k) \tag{1}$$

We must ensure that the following constraints are enforced:

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K} \sum_{j}^{j \in \mathcal{J}} {x_j}^i(k) = 1 \tag{2}$$

$$\forall j \in \mathcal{J}, \forall k \in \mathcal{K} \sum_{i}^{i \in \mathcal{I}} w_i(k) {x_j}^i(k) \leq W_j y_j(k) \tag{3}$$

$$\forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K} | h_i(k) = 0, {x_j}^i(k) = {x_j}^i(k+1) \tag{4}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, {x_i}^j(k) \in \{0,1\}, y_j(k) \in \{0,1\} \tag{5}$$

Equation 2 represents the obligation for a tenant to be placed at each time slot on an active cloud resource. Equation 3 means that the sum of the required capacity for each tenant on one cloud resource cannot exceed the capacity of the cloud resource. Equation 5 represents the variables we use. Equation 4 represent the migration strategy. The equality constraint means that for a tenant i and a time slot k, assignation values ${x_i}^j(k)$ will stay the same on time slots k and $k+1$. When a tenant is authorized to migrate between resources, there is no constraint for this tenant. Generalizing this on all resources produces the desired effect: tenants will be migrated from one resource to another only during the time slots specified by the migration strategy. The pre-defined migration strategy is symbolized here by the variable $h_i(k)$.

Finding cheap migrations strategies is primordial in our approach. This is the goal of our genetic algorithm. We present it in the following Section.

2.4 Cost optimization via genetic algorithms

A genetic algorithm is a well known meta-heuristic belonging to the family of evolutionary algorithms, and inspired by natural selection [4]. Its principle is to produce directed random evolutions on a population of individuals until it obtains one or several individuals with a good fitness value. Individuals are usually vectors of boolean values, whose corresponding fitness can be evaluated. Iterations are triggered until an end condition is reached.

We want to find the best migration strategy for all the tenants and time slots. Figure 3 shows our approach. The general principle is to use our iterative time slot heuristic (or the restricted model we have described in the previous Section) for evaluating migration strategies, until we find the best. To represent an individual, we vectorize a migration strategy by concatenating migration strategies of each $|\mathcal{I}|$ tenants (each one corresponding to a vector of D boolean values, $D+1$ being the number of studied time slots). The size of the vector will be $D \times |\mathcal{I}|$, with each element being equal to zero or one. For instance with two tenants and three time slots, the first migrating on the second time slot and the second tenant on the third time slot, we will have the following representation: $\big[0\ 1\ 0\ 0\ 0\ 1\big]$.

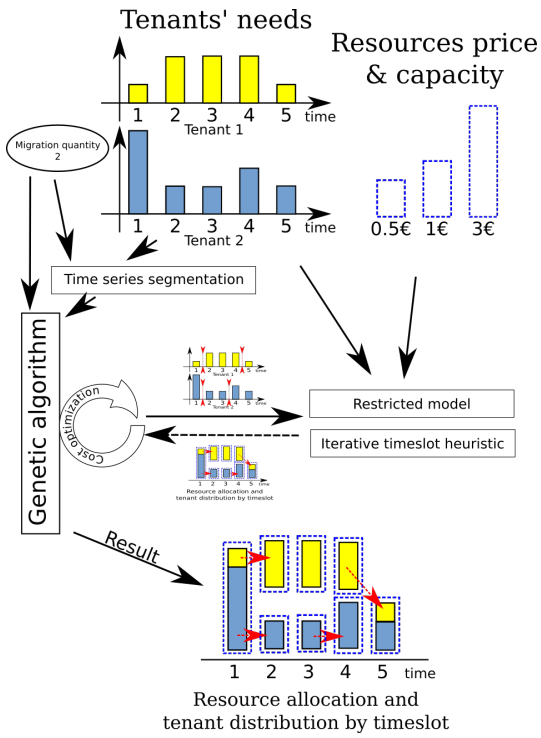

Fig. 3. A genetic algorithm to find better strategies

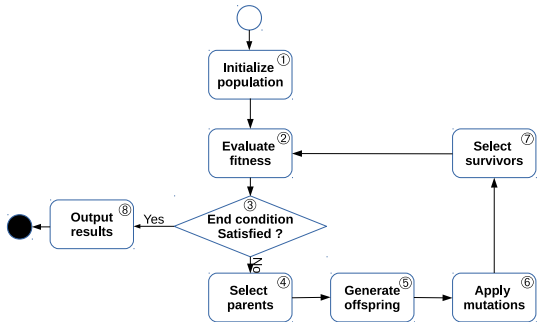

Fig. 4. Genetic algorithm phasis

The reader will find in figure 4 a brief description of the different steps of a genetic algorithm. Compared to the traditional approach, we have switched the mutation phase and the crossover phase. The co-hosted mutation requires to know the cost of the migration strategy in the population. We compute the cost in the fitness evaluation phase. The crossover phase generates potentially unknown (not yet computed) migration strategies. Thus, we do it after the mu-

tation phase. In our case, "parents" are mutated instead of the offspring. In the following, we describe the solution we designed.

Population initialization We initialize the population with several segmentation algorithm combinations (as we described in Section 2.2), and with random individuals with the correct number of migrations for each tenant.

Fitness evaluation We want to find the migration strategies that produces the lowest cost. The fitness score corresponds to the total cost of all the active resources on the time slots. To evaluate it on the different individuals, we compute the cloud resources allocation and placement of the tenants on it. In our case, we run our iterative time slot heuristic [2] or a solver on our model presented in Section 2.4, for each individual (migration strategy) we need to evaluate. We keep the cloud resources and tenants assignation distribution in memory for the next steps, and the fitness score.

Termination condition We use a time limit termination condition. This allow us to compare different solutions based on this limit.

Parent selection For this step, we use a classical rank selection strategy. We sort the population by fitness and we select randomly, and with a higher priority, the individuals with the higher rank (lower fitness or price in our case) for parents.

A specific mutation: co-hosted tenant migration mutation strategy In classic approaches, mutation updates randomly individuals, depending on a mutation rate, switching scalar values from zero to one or the other way around [4]. Here, the goal is to generate brand new individuals in the population, with non tested configurations. In our case, we cannot use a totally random approach, as the number of migrations for each tenant is bounded. However, even a randomized approach keeping a fixed number of migrations will not provide the desired effects as we can see on the left side of figure 5. We have noticed that most of the times, resources are not liberated as only one of their tenants is authorized to migrate. It limits the savings of resources liberation. We developed an alternative mutation more suited to our problem.

It consists in shifting the authorization to migrate for each co-hosted tenant at the indicated time slot for the reference tenant's resource. To achieve this goal, for each tenant, we browse the past time slots until we find an authorization to migrate or until we reach the beginning of the time slot space. If we find one, we set it to zero while setting to one the "destination" time slot. If the "destination" time slot is already set to one, we ignore this behavior. The example on the right side of figure 5 describes this principle. There, it is possible to migrate all the tenants of resource R1 to the cheaper resource R2, and thus reduce costs [1].

[1] The simple approach on the left of figure 5 keeps tenants on resource R1, it can never be freed

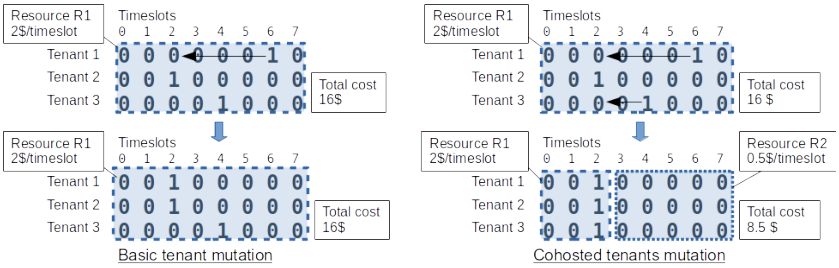

Fig. 5. Basic tenant mutation vs cohosted mutation

Specific offspring generation: the tenant crossover strategy The crossover phase consists in randomly mixing individuals (parents) of the current population in order to generate new individuals (children) having characteristics of both parents. The crossover technique that we use consists in switching the migration time of random tenants. First, two children identical to two migration strategies parents are generated. Then, depending on the number of tenants specified, each one will see its migration times switched in the children.

Generational replacement We replace the entire population with the offsprings, except for the best individuals from the original population (named elites). They replace the less fit offspring in the future population.

In the next Section, we present our experiments and the results.

3 Experimentation

We have conduct tests with the cloud resource prices and sizes, and the seeds of our previous work [2]. We consider 12 configurations, each composed of two Amazon Web Services compute resources: one database resource (RDS) for the database, and one compute instance (EC2) for the application server. Prices are comprised between 0.177 $ per hour for a BPM task throughput of 16.4 tasks per second, and 4.126 $ per hour for a BPM task throughput of 129.279 tasks per second.

For the customer part, we vary the number of tenants (10, 25, 50 and 100), and we use different throughputs in terms of BPM task per second. These throughputs are based on usage of anonymous customers of the BPMS Bonita[2]. We consider 6 configurations, needing a throughput respectively between 1 and 120, 14 and 16, 0 and 120, 1 and 3, 5 and 120, and 0 and 4[3].

We generate each tenant's initial time slot throughput randomly following an uniform distribution between the two throughputs. Our next step is to generate

[2] http://www.bonitasoft.com/

[3] The data and the results are available at: `https://doi.org/10.5281/zenodo.1173617`. The source code of the framework is not public, except for the segmentation library, available at `https://github.com/guillaumerosinosky/Segmentation/`.

the variation of throughput between time slots by adding or removing a random value limited to one quarter of the difference between the maximum and the minimum throughput. For our experiments, we used the Python library Inspyred [5] for the genetic algorithm that integrated well with our environment.

Experiment Parameters In order to obtain significant and realistic results, we used the following parameters:

- each test is launched for 10 different random seeds (i.e tenants' loads)
- a time slot size of one hour, as it was the reference duration of AWS cost model for computing instances at the time of the experiment.
- we choose to consider 4 migrations per day. A migration produces an interruption of around 10 seconds depending on the quantity of data.
- we consider a 2 days period (thus limiting migrations to 8, for 48 time slots).
- we consider the following parameters for the genetic algorithm: the number of elites individuals to 5, a mutation rate of 0.4, a population size of 20, a number of mutation points corresponding to the number of tenants divided by 5. These parameters were chosen following tests on multiple values for a limited number of seeds each. Details on this choice cannot be included for space restriction reasons.
- we limit the genetic algorithm computation to 600 or 1800 seconds and the solver computation time to 5 seconds.

3.1 Results

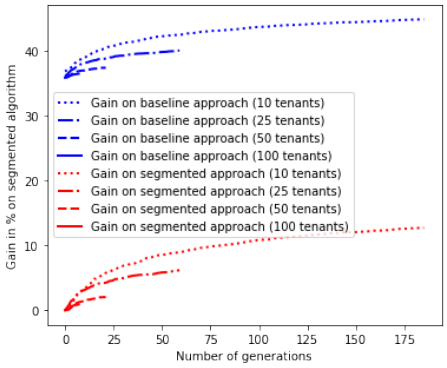

Fig. 6. Mean genetic algorithm gain on best initial segmented population for 600 second of running time

In figure 6, we show the relative gain of this approach compared to our previous approach (segmented approach) in red (in the upper part of the figure), and to a baseline approach in blue (in the lower part of the figure). The gain is better

for 10 tenants than for 100 tenants since the system has more time to search for the cheapest solution. For 10 tenants, we obtain more than 10 % enhancement compared to the previous approach, and more than 45 % compared to the baseline approach. However for 100 tenants, we have only a 1 % enhancement.

It appears that either the iterative usage of the heuristic, the genetic algorithm or the two of them is more efficient for a small number of tenants for the same number of generations. This is why we conducted experiments where we apply the proposition to subsets of the tenants and we aggregated the results as described in the next Subsection.

3.2 The splitting strategy

For this solution, we split the set of tenants into small groups selected randomly. We have tested different size of groups with various number of tenants and we applied the previous method keeping the same total computation time. Figure 7 shows the results we obtained with the genetic algorithm and the iterative heuristic. The x axis corresponds to the size of the groups of tenants. The y axis shows the relative gain compared to the results with no partition. A subset size with the same size as the number of tenants corresponds to no split, the gain is zero.

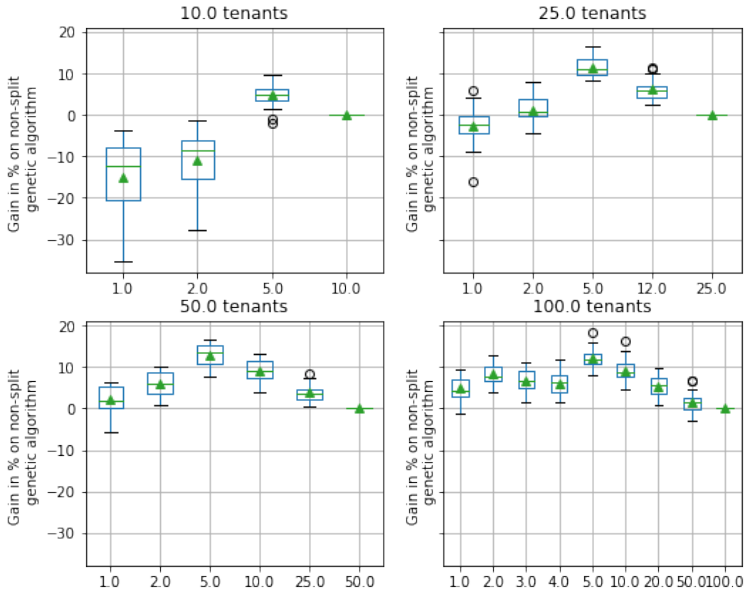

Fig. 7. Gain depending on splitting strategy for various split quantities.

We obtain the best results with partitions of 5 tenants in all cases. For the experiments we ran, the gain varies from 5% to 15%. We have no good expla-

nation for this result that we can reproduce. Our tests with the solver give the same results for the size of the groups as with the heuristic. In the next Section we present our results with groups of 5 tenants.

3.3 Results for solver and iterative heuristic

We implemented our model (presented in Subsection 2.4) using the optimization library PuLP [4] with the Gurobi solver[5]. For execution time and cost reasons, we were not able to test every set of parameters. For instance, with our current implementation, we managed to obtain results with the solver only up to a size of 25 tenants for the partition. Indeed, the duration of the initialization part and the required memory makes it impossible to run with more tenants. Thus we have limited our tests to parts of 5 tenants, for a total of 50 and 100 tenants. As we can see, the results stay close to the results of the heuristic. Figure 8 shows the absolute gain we obtained, and the corresponding percentage compared to the *baseline approach* cost, for 600 seconds and 1800 seconds of running time. We also present the non-split result for the segmented approach (results of the previous paper), and the split segmented approach where we apply time series segmentation on the groups of 5 tenants instead of all the tenants simultaneously.

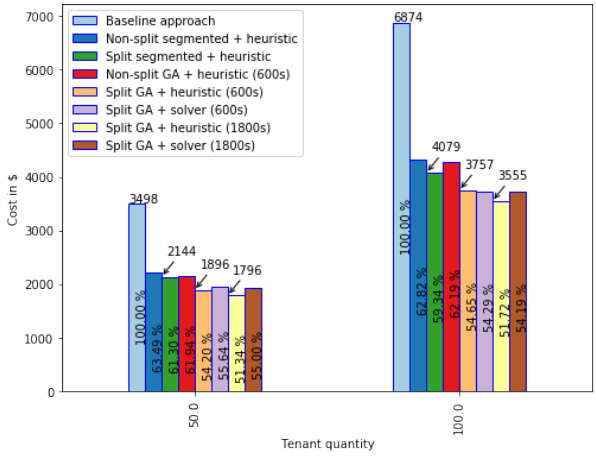

Fig. 8. Mean cost comparison for 50 and 100 tenants per group of 5

For 1800 seconds of execution time of the genetic algorithm, split heuristic give the best results. Mean distribution costs are 51.34 % for 50 tenants, and 51.72 % for 100 tenants compared to the cost of the intuitive approach. Using the solver gives good results but more expensive (respectively 55 % and 54.19

[4] `https://pythonhosted.org/PuLP/`
[5] `http://www.gurobi.com/`

%). For 600 seconds of execution time, the results are more balanced: they vary between 54.2% and 55.64%. The genetic algorithm does not enhance the results a lot for both approaches after 600 seconds: 3% for the heuristic and less than 1 % for the solver. Still, it enhances the initial split segmented results from 61.3 % to 51.34% for 50 tenants, and from 59.34 % to 51.72 % for 100 tenants.

We observe that the split segmented approach allows to gain more than 2 % , and to unleash the results of the genetic algorithm. Without splitting we gain around 1 % for 600 seconds of genetic algorithm compared to the original population (non split segmented). When splitting, the genetic algorithm results in a gain of 7.1 % for 50 tenants, and 4.69 % for 100 tenants compared to the split segmented strategy. The absolute gain compared to the intuitive approach remains worthwhile for 2 days: we save 1702 $ for 50 tenants and 3319 $ for 100 tenants for a cost of respectively 3498$ and 6874$. The respective gain compared to our previous work is 425 $ and 763 $.

4 Related work

Many researchers have studied elasticity in the cloud and elasticity for BPM or orchestration systems. Schulte and al. [6] did a general review on the topic and gave directions for future research. In this paper, we focus on the resource allocation and scheduling problem and use a tenant-centric approach based on BPM task throughput, instead of the BPM process-centric from other approaches. Rekik et al. [7] propose an integer programming model based on general hardware metrics for BPM elasticity on the cloud. They base their approach on resource allocation and BPM task scheduling. They do not consider multiple time slots, tenant migration or multi-tenancy. Other attempts on BPM elasticity in the cloud exist such as [8], [9], [10]. Though not cloud-related, Djedovic et al. propose in [11] a genetic algorithm for BPM task scheduling to their corresponding resources. It uses a representation of each resource. They want to minimize the waiting time and the global resource cost. Junhke et al. [12] propose a task focused genetic algorithm for BPEL workflows scheduling in distributed Clouds.

On other subjects than BPM, the machine reassignment problem [6] is close to our problematic. It considered software reassignment problem on virtual machines including the migration cost. Gavranovic et al. [13] obtained the better results to this challenge. However, this problem is based on hardware metrics, and aggregate migration cost in the objective function. Our problem is not exactly virtual machine allocation since the hardware is already defined. Numerous other attempts target virtual machines, such as [14]. Automated approaches based on cloud offers retrieval and hardware requirement for software such as [15] are also valuable.

These works do not consider simultaneously multi-tenancy, multiple instance types, and migrations, except in the form of data transfer cost for [10] or aggregated migration cost for [13]. In this paper, we present an evolution of our

[6] `http://www.roadef.org/challenge/2012/en/`

previous work [2]. We have based our approach on time series segmentation for deducing the "good" time slots to migrate tenants, and on the iterative use of an enhanced version of our time slot heuristic[16]. We also presented the corresponding ILP (Integer Linear Programming) model. Results were encouraging compared to a baseline approach, but could be improved regarding the results that we obtain with a solver. We could not compare with other approaches since most of them do not consider migrations of data as an issue. They scale up by adding compute resources to the process engine, considering that access to the database is not a problem. From our experience, at some point, the database is always a bottleneck.

5 Conclusion

In this paper, we proposed a method for cost optimization of BPMaaS deployment based on tenant migration strategies and a genetic algorithm. We presented a new integer programming optimization model. Both allows to obtain substantial gains for BPMaaS providers. The result we obtain when we group the tenants is interesting. It may be explained by the size of the objective space. The fact that it is reproducible for different number of tenants shows that testing multiple sizes may allow providers to save on the operation cost. Moreover, using other metaheuristics such as simulated annealing or hill climbing could provide even better results.

Our method was tested with BPM task throughput but could work with other metrics that can be expressed as a scalar for both the cloud resources and the tenants. We can consider for instance the number of processes, or the number of HTTP requests that lead to transactional processing. Our methods can then be generalized on systems non related to BPMS using multi-tenancy and tenant-related persisted data. A BPMS execution engine behaves more or less like a transactional web application. Our approach is offline and require to anticipate on the tenant load. For many business cases, this is a valid assumption. The server load is relative to the number of employees and the number of cases they can execute everyday or hour with little variations. A next obvious step would be to couple our algorithm with prediction systems. This would provide an effective online algorithm. It could adapt to unforeseen variations.

6 Acknowledgment

The authors would like to thank Gurobi for the usage of their optimizer, and Amazon Web Services for the EC2 instances credits (this paper is supported by an AWS in Education Research Grant Award).

References

1. S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi, "Live database migration for elasticity in a multitenant database for cloud platforms," CS, UCSB, Santa Barbara, CA, USA, Tech. Rep, Tech. Rep., 2010.

2. G. Rosinosky, S. Youcef, and F. Charoy, "Efficient Migration-Aware Algorithms for Elastic BPMaaS," in *Business Process Management*, J. Carmona, G. Engels, and A. Kumar, Eds. Cham: Springer International Publishing, 2017, vol. 10445, pp. 147–163.
3. M. Lovrić, M. Milanović, and M. Stamenković, "Algoritmic methods for segmentation of time series: An overview," *Journal of Contemporary Economic and Business Issues*, vol. 1, no. 1, pp. 31–53, 2014.
4. D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.
5. A. Garrett, "inspyred: Bio-inspired Algorithms in Python — inspyred 1.0 documentation," 2014.
6. S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic Business Process Management: State of the art and open challenges for BPM in the cloud," *Future Generation Computer Systems*, 2014.
7. M. Rekik, K. Boukadi, N. Assy, W. Gaaloul, and H. Ben-Abdallah, "A Linear Program for Optimal Configurable Business Processes Deployment into Cloud Federation." IEEE, Jun. 2016, pp. 34–41.
8. S. Euting, C. Janiesch, R. Fischer, S. Tai, and I. Weber, "Scalable Business Process Execution in the Cloud," in *2014 IEEE International Conference on Cloud Engineering (IC2E)*, Mar. 2014, pp. 175–184.
9. J. Xu, C. Liu, X. Zhao, S. Yongchareon, and Z. Ding, "Resource Management for Business Process Scheduling in the Presence of Availability Constraints," *ACM Transactions on Management Information Systems*, vol. 7, no. 3, pp. 1–26, Oct. 2016.
10. P. Hoenisch, C. Hochreiner, D. Schuller, S. Schulte, J. Mendling, and S. Dustdar, "Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds." IEEE, Jun. 2015, pp. 17–24.
11. A. Djedović, E. Žunić, Z. Avdagić, and A. Karabegović, "Optimization of business processes by automatic reallocation of resources using the genetic algorithm," in *Telecommunications (BIHTel), 2016 XI International Symposium on*. IEEE, 2016, pp. 1–7.
12. E. Juhnke, T. Dornemann, D. Bock, and B. Freisleben, "Multi-objective Scheduling of BPEL Workflows in Geographically Distributed Clouds." IEEE, Jul. 2011, pp. 412–419.
13. H. Gavranović, M. Buljubašić, and E. Demirović, "Variable Neighborhood Search for Google Machine Reassignment problem," *Electronic Notes in Discrete Mathematics*, vol. 39, pp. 209–216, Dec. 2012.
14. J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11. New York, NY, USA: ACM, 2011, pp. 229–238.
15. J. M. García, O. Martín-Díaz, P. Fernandez, A. Ruiz-Cortés, and M. Toro, "Automated Analysis of Cloud Offerings for Optimal Service Provisioning," in *Service-Oriented Computing*, M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, Eds. Cham: Springer International Publishing, 2017, vol. 10601, pp. 331–339.
16. G. Rosinosky, S. Youcef, and F. Charoy, "An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing Environment." IEEE, Jun. 2016, pp. 311–318.