Efficient migration-aware algorithms for elastic BPMaaS

Guillaume Rosinosky[1 2], Samir Youcef[2], and François Charoy[2]

[1] Bonitasoft, Grenoble, France,
guillaume.rosinosky@bonitasoft.com,
http://www.bonitasoft.com
[2] Inria Nancy Grand Est - Université de Lorraine - CNRS

Abstract. As for all kind of software, customers expect to find business process execution provided as a service (BPMaaS). They expect it to be provided at the best cost with guaranteed SLA. From the BPMaaS provider point of view it can be done thanks to the provision of an elastic cloud infrastructure. The provider still have to provide the service at the lowest possible cost while meeting customers expectation. We propose a customer-centric service model that link the BPM execution requirement to cloud resources, and that optimize the deployment of customer's (or tenants) processes in the cloud to adjust constantly the provision to the needs. However, migrations between cloud configurations can be costly in terms of quality of service and a provider should reduce the number of migration. We propose a model for BPMaaS cost optimization that take into account a maximum number of migrations for each tenants. We designed a heuristic algorithm and experimented using various customer load configurations based on customer data, and on an actual estimation of the capacity of cloud resources.

Keywords: BPM, cloud, elasticity, BPM as a service

1 Introduction

During the last decade, we witnessed a major change in the way companies are delivering software. It is more often distributed as a service, operated by software producers, hosted in public clouds instead of as a package and installed install on premises. BPM systems vendors and operators start to propose this distribution modality. It removes the burden for customers to operate the BPMS and the corresponding infrastructure. They pay for process instances they execute or on a fixed monthly rate per user [1]. The service provider aims at providing the required service quality at the lowest possible cost. Thanks to the public cloud and the elasticity it supports, providers can deliver that quality while minimizing resource consumption, and thus the operational cost. Public cloud providers allow to add and remove dynamically computing resources. However, it does not fit well with the deployment stack of a BPMS that include web servers and databases. In this paper, we propose a method that allows to distribute process

execution on a set of cloud computing resources, and to adjust the resources based on the load that the customers require. We call a customer *a tenant*, and we ensure that all the processes of a tenant are executed on a BPMS installation. We consider that resources are paid by discrete time units. We assume that we know for each tenant what will be its maximal resource consumption time unit per time unit. More precisely, we want to take into account the knowledge we can get from the business dimension, i.e. the number of task execution per hour. In our previous work [2], we proposed to optimize resource consumption from a timeslot to the next. In this paper, we extend the method to optimize resource consumption on an arbitrary number of timeslots. We also limit tenant migrations from an installation to another to avoid unwanted service disruption. Migrations generally occurs when the resource on which a tenant is deployed is not sufficient to support its required load for the coming period of time. For each studied duration, based on the knowledge of the resource consumption profile that we can get from all the tenants, we compute a deployment plan that minimize resource consumption while maintaining the number of migrations for each tenant at an acceptable level. This is the contribution of this paper. We propose a linear optimization model and then an heuristic in two parts. A first that computes good times for each tenant to migrate using time series segmentation methods. Then we show how this method coupled to the a restricted version of the timeslot heuristic from the previous paper provides substantial gains compared to more simple or naive ones. We validate it with an experimentation using realistic values for the size customers and the size and price of cloud resources. In the next section, we present the state of art regarding BPM elasticity in cloud computing field. In section 3 we describe the model that we want to optimise. In the following section (4), we explain the condition of the experimentation. Then in section 5 we describe and discuss our results. In the last section, we conclude and present possible extensions to this work.

2 Elasticity in BPM

Some attempts have already been done at managing BPMaaS elasticity in the cloud. Schulte et al. [3] made a review on the current status on BPM elasticity, and the different important criteria. We focus here on the scheduling and resource allocation parts, with an emphasis on the multitenancy functionality.

Hoenisch et al. [4] proposes an interesting approach : the Service Instance Placement Problem, a cost optimization model concerning the assignment of process instances to VM, scheduling of service invocations, and the provisioning of VM. The authors consider the underline structure of the BPM processes and propose to optimize the cost while taking into account penalties for violated deadlines. However, they consider only the BPM engine's CPU and RAM capacity in their model - and not the database tier- and do not consider multitenancy or migrations. Other previous attempts have the same drawbacks such as [5], [6], [7].

Hachicha et al. [8] addresses multi-tenant BPMaaS with the concept of configurable resource assignment operator. It consists of an enrichment of the process with meta informations on the required resources for the tasks execution. However, it needs to add informations in the BPM schema, thus needing to alter BPM engine and the processes of the customers and do not propose a resource allocation and scheduling method. Another attempts to tackle the multi-tenant problem is done in [9] by Sellami et al. They propose a multi tenant approach based on customizable thresholds, however, it does not take into account migration cost or the database tier.

There are many papers on Virtual Machine assignment or reassignment in data centers such as [10], [11], [12], [13], however as in datacenters the physical machines are already reserved, only the scheduling part is studied and for the reassignment papers, the migrations are usually counted as part of the objective function instead of a separated entity.

Our work is an evolution of our precedent paper [2] where we proposed an bi-objective optimization model for cost and migrations quantity for all tenants scrambled from a timeslot to the next, and a corresponding efficient heuristic. Simply repeating this heuristic on multiple timeslots could provoke the migration of the same tenants, thus provoking breaks of quality of service. We propose here to harness the problematic of optimization on multiple consecutive timeslots, with a limitation on the number of migrations for each tenant.

3 The BPM execution model

In this section we introduce the model for the BPMS execution we want to optimize. Our model relies on a few assumptions regarding the BPM system. First, it must be multi-tenant, i.e. several customers (or tenants) can share the same BPMS installation. Second, it is possible to migrate a tenant from one installation to another with minimal disruption. The main issue here is the data migration from a database to another [14, 15]. We also only consider that IaaS providers bill the computing resources per studied timeslot (for instance per hour). Main public cloud providers like AWS, and IBM Bluemix, follow this pattern per hour, while Google Compute Engine or Azure propose also a per minute billing.

The operation of a BPMS requires a complex production software stack. It may combine a BPM engine, load balancers and relational databases. They are often deployed on distinct hardware instances or virtual machines from the BPM engine, mostly for performance reasons. We call "*cloud configuration*" the set of resources that we use to execute process instances for a group of tenants. Last, we assume that we know the usage requirement for each tenant timeslot by timeslot in term of the maximum task throughput per second. This metric has the advantages of being representative of the system usage of both the database tier and the application server tier. We must also know the capability of each cloud configuration type in term of BPM task throughput.

We aim at minimizing the cloud resource cost while ensuring that the throughput for each tenant is at the required level. Migrations count for each tenant should not exceed a fixed number in order to avoid disruptions. We propose here a linear model where we wish to optimize the cost of placement.

Let the following variables :

- $\mathcal{T}$, the set of cloud configuration 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 configurations associated with each tenant. its cardinality is $m = t \times n$
- C_j, and W_j, respectively the cost and the capacity for the configuration j, with j in $\mathcal{J}$
- w_i(k), the required capacity 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 is the defined maximum number of migrations for each tenant

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

We have the following constraints :

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

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

$$\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {x_j}^i(k) {x_j}^i(k+1) \geq |\mathcal{K}| - M \tag{4}$$

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

Equation 1 is our optimization objective. We want to minimize the total cost of cloud configurations for all the time slots (a day for instance). The constraint described in equation 2 means that, for each timeslot, each tenant must be located on one and only cloud configuration. The constraint described in equation 3 means that, for each timeslot, the sum of required throughput of the tenants co-located on a cloud resource do not exceed the capacity of this resource.

The constraint described in equation 4 means that we want to limit the number of migrations per tenant to M. If ${x_j}^i(k)$ and ${x_j}^i(k+1)$ are both equal to 1 for the resource j and the tenant i on two consecutive timeslots k and $k+1$, their product will be equal to 1, the tenant did not migrate. In the other cases, the product will be equal to 0. They occur when tenant i migrated from or to

another resource from timeslot k to timeslot $k+1$ or when tenant i remained on a different resource on both timeslots. We sum these products resource per resource, on each timeslot pair for each tenant. We obtain the number of timeslot where a tenant remained on the same configuration. The difference between the total number of timeslots and this number is the number of migration. Limiting the number of migrations is then straightforward.

Since we have multiplication between two variables, we obtain a quadratic optimization problem. As can become is very slow to compute, we linearized the equation 4, following the usual method. The result for this linearization is described in equation 6 :

$$
\begin{aligned}
&\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {w_j}^i(k+1) \geq |\mathcal{K}| - M \\
&\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {x_j}^i(k) + {x_j}^i(k+1) - 2{w_j}^i(k+1) \leq 1 \\
&\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {w_j}^i(k+1) \leq {x_j}^i(k) \\
&\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {w_j}^i(k+1) \leq {x_j}^i(k+1) \\
&\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, {w_i}^j(k) \in \{0,1\}
\end{aligned}
\tag{6}
$$

Even with the linearization, the resolution of this problem can be very time consuming. The number of variables will be of $tn(D+1) + tn^2(D+1)$, and the number of constraints will be of $tn(D+1)+tn^2(D+1)+4n$. For 7 cloud resource types(t), 100 tenants (n), and 24 time-slots ($D+1$), it makes 1696800 variables, and 1697200 constraints. It is not reasonable to try to compute the optimal solution when the number of tenants grows, as we will see in the experiment part. In the next section, we propose a heuristic that provide solutions to the problem with reasonable computation time even with large number of tenants.

4 Heuristic optimization proposition

4.1 Iterative timeslot algorithm

This algorithm is based on our previous timeslot heuristic [2]. Its principle is to consider that, regarding an initial distribution, we search the best distribution for the next timeslot, knowing that the required capacity of each tenant can change. With the timeslot algorithm, we search for a Pareto front of the lowest global number of migrations and resources cost. As the number of migrations is discrete and limited by the number of tenants, we can compute the lowest cost for each number. First, we look at overloading and overloaded tenants as shown in figure 1.

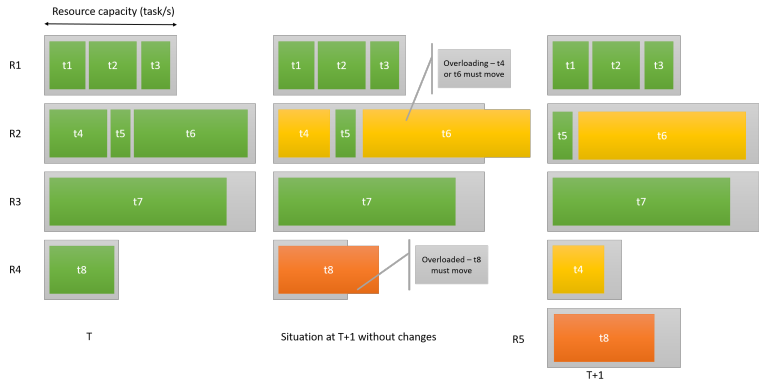

Fig. 1. Example of distribution of tenants on cloud resources at T and T+1

It depicts the distribution of tenants on different cloud configurations. The initial state is at time T. At time $T+1$, the requirement for each tenant changes. Some of them have to migrate (orange and red). The heuristic principle is, that for each possible number of migrations we consider the combination of resources containing the corresponding number of tenants, added to the number of tenants that we must migrate. As we consider the total quantity of tenants, there are several possibilities for each number of migrations. This is a classic subset sum problem. For instance, in figure 1, in order to compute the results for 5 migrations, we should move the tenants $t8$, $t6$, and remove the combinations of resources with 3 remaining tenants ($R1$ or $R2$ and $R3$). Once we have selected the tenants, we first repack the possible tenants in existing resources, and then use a Variable Cost and Size Bin Packing [16] algorithm for the remaining ones. The last step consists by iteratively try to replace by cheaper ones the resources with only moved tenants.

This approach provides good results timeslot by timeslot. Still, we must adapt it to ensure the number of migrations per tenant described by constraint 4 that limit the number of migration for a given duration per tenant. In the next section we propose an adaptation of this algorithm to enforce this constraint.

4.2 A migration aware optimization strategy

For this new strategy, we add the list of tenants allowed to migrate as an additional parameter to the previous method. For instance if, for a timeslot, we allow to migrate every tenant but $T5$ and $T2$ because they have reached their maximum number of migration (M), the new timeslot algorithm must ignore them and maintain them on their resources. We cannot delete resources with tenants.

Let a *migration strategy* the set of $h_i(k)$ with $0 \leq k \leq D-1$ where each tenant i is allowed to be migrated. $h_i(k)$ is equal to 0 if the tenant is not allowed to move between timeslot k and $k+1$, and equal to 1, if it is allowed. The

equation 7 describe the maximum number of migrations.

$$\forall i \in \mathcal{I} \quad \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} h_i(k) = M \tag{7}$$

We need to find the optimal values for each $h_i(k)$ respecting the maximum number of migrations to obtain the best cost.

Once we have determined the different migrations timeslots, we choose the required capacity level. As our algorithm does not consider multiple timeslots simultaneously, we assign a fixed capacity for each tenant for each period where it does not migrate. We call $C_i(m)$ the capacity during the period between migrations m_1 and m_2. In order to avoid overloads, we consider the maximum capacity required for the corresponding timeslots, as in equation 8. An example of a maximum load strategy is presented in figure 2. These capacities are used instead of the initial capacities of the tenants in the timeslot algorithm.

$$\forall i \in \mathcal{I} \forall m_1 \leq k < m_2, C_i(m_2) = \max_{m_1 \leq k < m_2} (w_i(k)) \tag{8}$$

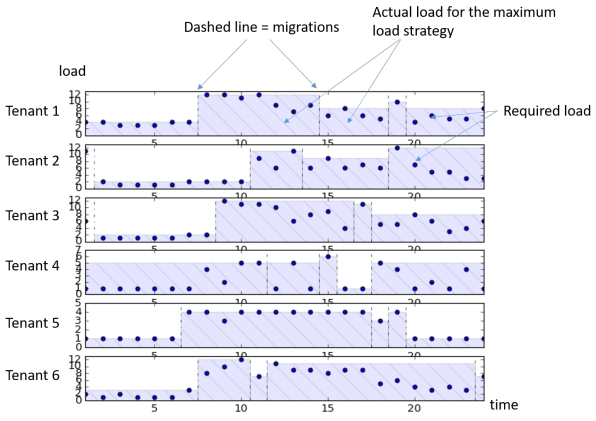

Fig. 2. Migration strategy of 6 tenants on 24 hours

As testing every migration strategy possibilities would be require too much time, we need an efficient way to evaluate which one we should use. It should give better results than a naive approach, and respect the constraint on the number of migrations for each tenant.

4.3 Time series segmentation

We propose a method to identify good migrations strategies. We consider first that the variations in load will produce the need for migrations. If for instance a tenant needs a throughput of 10 tasks per seconds between 12pm and 6am, and then a throughput of 50 tasks per second between 6am and 12am, the best time to migrate is at 6am. Our approach here is, for k migrations, to find a way to fragment the load time series in $k+1$ consecutive fragments, in a way where

they have the minimum load. Time series segmentation techniques address this kind of problems.

As Lovric[17] explains, we can see time series segmentation as a processing step and core task for variety of data mining tasks, as a trend analysis technique, as a discretization problem in function of dimensionality reduction, etc... The latter point interests us as we want to find a way to discretize the load time series, with discrete periods of remaining tenants, separated by migrations. The main common algorithms based on Piecewise Linear Representation are originally reported by Keogh et al. [18] : top-down, bottom-up and sliding window. As our approach is offline (we know the future load), we focus only top-down and bottom-up.

The principle is to iteratively separate (top-down) or merge (bottom-up) consecutive sets of observations in the time series, so they keep a minimal error related to real observations. As it can be seen in [18] and [17], the main version of the algorithm segments the time series using Piecewise Linear Approximation (PLA), fitting each segment with an affine function found by linear regression of the values for each segment. This approach is interesting for our needs. Indeed, as we explained earlier, we want to segment the time series considering the maximum load instead of the mean of segment (we consider for each segment its maximum load as described in equation 8). As we will see in the experiment section, we also tested a mean constant piecewise approximation, that consider a fixed mean for the segmentation.

Once we obtain the segments, we compute the corresponding migration strategy. We initialize the matrix to zero, except for the timeslots where there is a change of segments, that we initialize to one. We then use this migration strategy with the restricted iterative timeslot algorithm as we can see in the next part.

4.4 The optimization algorithm

Here is a synthesis of our timeslot algorithm :

- First, compute the desired migration strategy using a time series segmentation algorithm.
- Second, initialize the initial timeslot (zero) with the initial distribution.
- Then, for each timeslot from $k = 1$ to $k = D$:
  - Launch a timeslot algorithm, using the distribution of the previous timeslot. Only the tenants able to move in the migration strategy for this timeslot migrate. Thus we ensure constraint 4 of the model
  - Keep the less expensive distribution with the least number of migrations. This is the distribution we choose for the current timeslot.

This algorithm provides a solution that enforces the constraints. In the next section we describe our experimentation that shows how it provides better results than a naive solution and with a reasonable computation time even for an hundred of tenants.

5 Experimentation

To test our solution we made a few assumptions and relied as much as possible on datasets that gives us realistic foundations for the resolution of the model. As we can see in the model part, we needed to have a good estimation of the customer loads, timeslot by timeslot on one side, and of the price and capability in task throughput per timeslot of a cloud configuration on the other side. We have then compared several segmentation methods on the same datasets of customer loads and cloud configurations to find the best ones. In the next part we describe the datasets.

5.1 Datasets

In order to get meaningful cost and task throughput for our cloud configurations, we have used the data obtained in our test framework experimentation [19]. In this paper, we have set up an experiment on AWS with a BPMN process composed of 20 consecutive automated tasks launching a Fibonacci script. We launched tests on several r3 (storage-oriented) family instance types for the database, and c4 (CPU-oriented) family instance types for the application server. We used the BPM system BonitaBPM 7.3.2 [3] in its Open Source version. We compared the obtained throughput with the price of each cloud configuration. Results are described in table 1. It provides the number of tasks for one $ for each configuration type.

DB inst. type	AS inst. type	price	task TP	task TP per $
db.m3.medium	m3.medium	0.177	16.400	92.656
db.m3.medium	c4.large	0.223	23.157	103.845
db.r3.large	c4.large	0.399	55.164	138.255
db.r3.large	c4.xlarge	0.518	58.067	112.100
db.r3.xlarge	c4.large	0.674	65.113	96.607
db.r3.large	c4.2xlarge	0.757	61.474	81.208
db.r3.xlarge	c4.xlarge	0.793	83.236	104.963
db.r3.xlarge	c4.2xlarge	1.032	89.149	86.384
db.r3.2xlarge	c4.2xlarge	1.587	105.794	66.663
db.r3.2xlarge	c4.4xlarge	2.063	107.585	52.150
db.r3.4xlarge	c4.4xlarge	3.173	115.283	36.332
db.r3.4xlarge	c4.8xlarge	4.126	129.279	31.332

Table 1. Price, mean task throughput, and mean task throughput by dollar of used cloud configurations.

For the customer load part, we wanted to test multiple tenant quantities (5, 10, 25, 50 and 100), having different throughputs based on real data from BonitaBPM customers. More precisely, we used minimum and maximum throughput per second found in the anonymized execution history tables. The used thresholds are described in table 2. We have then generated each tenants initial timeslot load randomly following an uniform distribution between the two thresholds.

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

customer	*days*	*minimum*	*maximum*
A	4	1	120
B	1	14	16
C	45	0	120
D	7	1	3
E	45	5	120
F	550	0	4

Table 2. For each customer, the observed interval in days, the minimum and the maximum task throughput per second for each hour.

To avoid too much variation between timeslots, we also used another parameter we name *tenant gap*, a percentage of the gap between a tenant's minimum and maximum throughput. Using a totally random behaviour makes tenants throughput very chaotic. In general, the required load is relatively stable, as we have noticed in customers data. For each timeslot, we compute randomly the percentage of the gap, and we add it to the previous timeslot's load. If we obtain a load lower than the minimum, we cap it to the minimum (respectively capped to the maximum for loads superior to the maximum). For instance, for customer E and a gap of 0.25, the change between hours would follow an uniform distribution between $-0.25(120-5) = -28.75$ and $0.25(120-5) = 28.75$, capped on the minimum and maximum for the customer, here respectively 5 and 120. We have experimented with various values for the gap. A gap percentage of 1 will correspond to a complete random behavior between the minimum and maximum loads. A gap percentage of 0 will correspond to a load who stays fixed at the initial timeslot load. For each number of tenants, we wanted to test multiple tenant gaps as it shows multiple levels of variability. More precisely, we have tested the tenant gaps 0.25, 0.5, 0.75 and 1. We have not tested the 0 tenant gap, because it provokes a stable load, and the algorithm would converge to a stable distribution and then stop moving tenants.

As we said, the main parameters we varied are the tenant quantity, and the tenant gap. Nonetheless, given the random nature of the obtained load, we ran the tests multiple times with different random distributions for each couple tenant gap and tenant quantity. In order to keep repeatable configurations and be able to test multiple algorithms, we have used twenty different random seeds for each tenant parameter couple. Using the same random seed on a same couple tenant gap/tenant quantity gives every time the same load distribution.

For each generated set (the cartesian product between twenty seeds, the 4 tenant gaps, and the 5 different tenant quantities), we tested several migration plan algorithms that we describe in the next part.

5.2 Software and methods

Our goal here is to determine between the different methods and for different number of migrations, which one gives the better migration strategies i.e. which one between top-down and bottom-up algorithms, grouped and individual strategies, and ConstantMaxPieceWise, ConstantPieceWise and Linear Regression fitters gives the best results.

As our metrics are hourly based on the different configurations and tenant loads, we used hourly timeslots, more precisely 48 timeslots. The initial setting is the following : for each tenant, we select the least expensive resource that is suited for the maximum required throughput on the study time. We name this method *adapted heuristic*.

For the segmentation part we implemented a fixed and updated version of the Alchemyst library[4]. This library implements top-down and bottom-up algorithms, with mean constant and linear regression fitters. We added max constant fitter as described in section 4.3. We also tested the algorithms for several number of migrations, more precisely 2, 3 and 4 per day, so 4, 6 and 8 for the 48 hours studied. Last, we tested two grouping strategies : considering each tenant in a separated manner (*individual strategy*), or executing the segmentation on the sum of the loads, and moving every tenant simultaneously at the obtained migration timeslots (*grouped strategy*) .

We also enhanced our previous timeslot algorithm implementation [2] with the restriction on the tenant list. For performance reasons we don't consider here the subset sum for each number of migrations but only all the tenants.

To obtain reference values, we used a naive method to compute the cost. This gives us a reference cost that we can obtain without calculation. We also compared a subset of our test dataset with the solving of the linear model described in chapter 3 during a limited time, for the lowest number of tenants. For this we used the solver Gurobi [21].

Table 3 summarize the different parameters we used in this experiment. Experiments have been executed on c4.xlarge (CPU optimized) instances on Amazon Web Services. In the next part, we discuss our results.

group	variable	size	values
data	tenant gap %	4	0.25,0.5, 0.75,1
data	seed number	20	-
data	number of days	1	2
data	number of tenants	6	5,10,25,50,100,200
data	number of migrations	3	2,3,4
segmentation	algorithm	2	bottom-up,top-down
segmentation	fitter	3	mean constant,max constant,linear regression
grouping strategy	tenant load	2	grouped,individual
timeslot algorithm	subset sum size	1	1

Table 3. Synthesis of the experiment dataset and algorithm used values.

5.3 Results and discussion

As we explained in the previous part, we compare our results with the *adapted heuristic* cost we obtained (in figure 3). The mean cost is between 370.15 $ for 5 tenants and a tenant gap of 0.25 and 18853.56 $ for 200 tenants and a tenant gap of 1. We can see here that the higher the gap, the higher the cost. Since the variation of the load is less restricted, the maximum load can be higher, and the strategy principle is to consider each tenant's higher load for each cloud resource.

[4] https://github.com/alchemyst/Segmentation developed by Carl Sandrock for his paper [20]

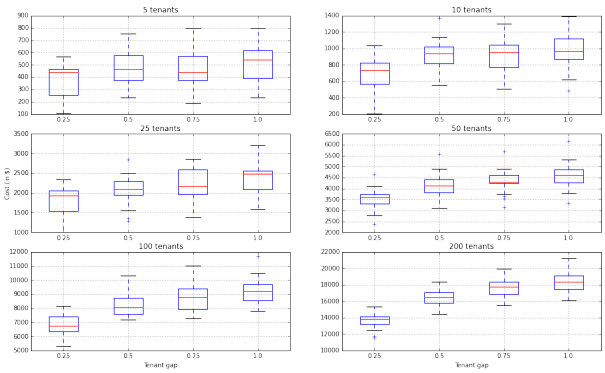

Fig. 3. Distribution of experimentation adapted heuristic costs in dollars

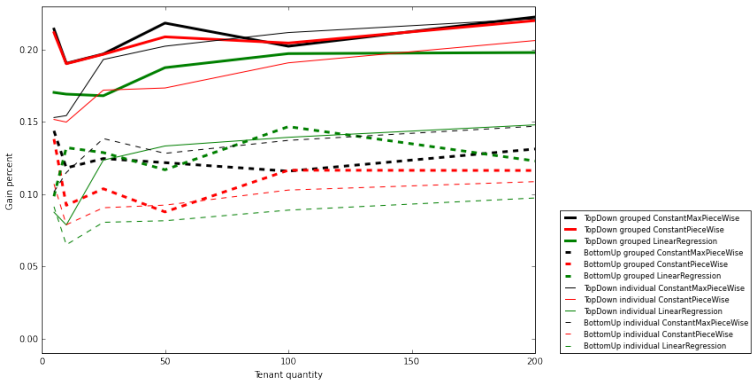

Fig. 4. Gain in percentage of the algorithm regarding the naive approach, compared to the number of tenants, for a tenant gap of 0.25 and 4 migrations per day.

We see in figure 4 the results we obtained for a tenant gap of 0.25 and 4 migrations a day with the heuristic. We compute the gain percentage by observing for each experiment run (one seed, one tenant gap percent, on tenant quantity, one number of migrations) the ratio of the difference of the result related to the corresponding adapted cost. We can see that the different strategies give different level of results. Top-down algorithms give better results than top-down, except for top-down individual with linear regression fitter. Grouped strategies give almost every time the best results, except for the top-down algorithm on individual strategy with a constant maximum piecewise fitter, who gives results near to the two bests, (top-down grouped constant strategies), and is even more efficient for 200 tenants. For top-down algorithms, constant maximum piecewise is the best algorithm, followed by constant mean piecewise. It is more difficult to compare for bottom-up strategies.

Figure 5 shows a global overview of the results. Most of the time, the top-down algorithm gives better results than the bottom-up. The best global approach are usually more efficient on small number of tenants or with less varia-

tion (low tenant gap). The best fitters here are top-down grouped constant max piecewise, top-down individual constant max piecewise and top-down constant piecewise. As expected, we obtain better results when we allow more migrations (from 5 to 10 percent for 2 migrations to 15 to 25 percent for 4 migrations). A higher gap percentage generates worse results. The top-down individual Constant Max Piecewise strategy gives the best results for more than 5 tenants almost every time.

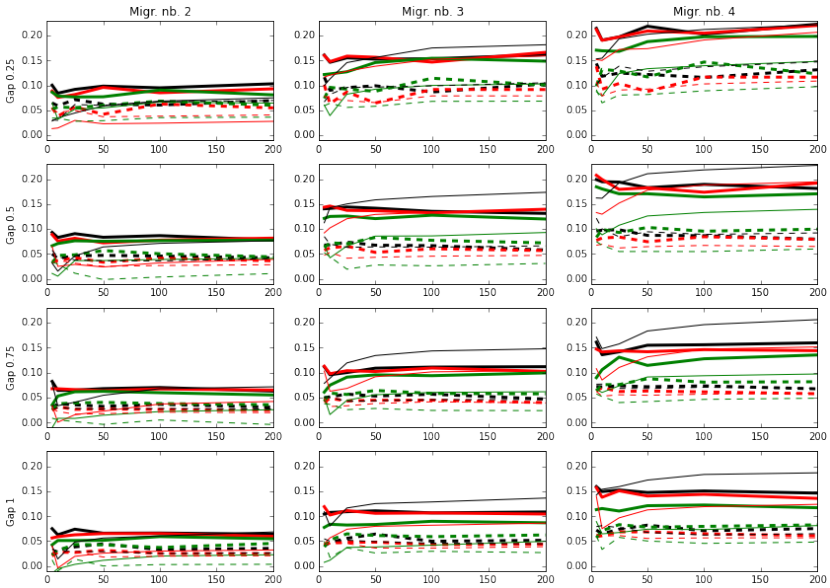

Fig. 5. Gain in percentage of the algorithm regarding the naive approach in y axis, compared to the number of tenants in the x axis. The legend of figure 4 applies here.

Figure 6 show the running time of the algorithm with different parameters. It stays relatively stable, for a defined strategy and number of tenants, mainly for grouped strategies. Individual strategies are always longer, and the duration seems to be multiplied by 3 to 4 each time the number of tenants doubles.

On the other side, computing the results with the exact model using a solver is very time consuming and does not give very good results for more than 5 tenants and a defined time of 30 minutes as we can see in the table 4. We have not tried to do the complete tests for more tenants. Even for 10 tenants, 24 hours of computing time does not give the optimal results : we have obtained a MIP gap of 7 % at best with the solver, while with our heuristic we obtained gains of 20% with a mean duration of at most 0.4 seconds of running time. For 100 tenants, our heuristic duration is 100 seconds for a grouped strategy.

We can see the correlation between the number of migrations and the gain. The results show us that the top-down algorithm works much better than the bottom-up, and that constant maximum piecewise fitter give almost every time better results than the other fitters. The constant mean piecewise fitter gives

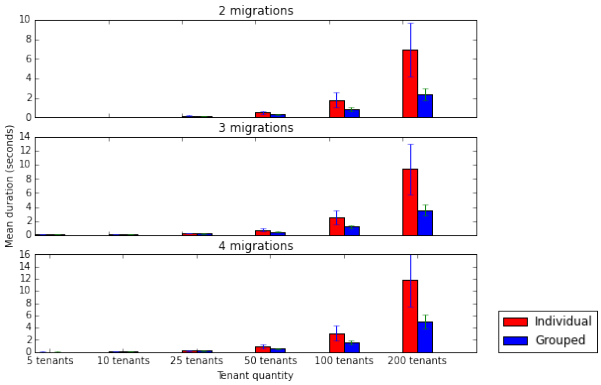

Fig. 6. Mean duration in seconds for individual and grouped approaches, for each studied number of tenants. The standard deviation is represented on each bar, with the mean number of seconds.

Tenant qty	Tenant gap	Mean adapted	Nb migr.	Solver duration	Solver gain	Mean MIP gap
5	0.25	419.99	2	1800	41.05%	3.57%
5	0.25	421.91	3	1800	48.28%	3.45%
5	0.25	421.91	4	1800	51.64%	3.91%
10	0.25	685.82	2	1800	3.18%	48.33%
10	0.25	685.82	3	1800	1.28 %	52.68%
10	0.25	685.82	4	1800	1.28 %	55.33 %

Table 4. Solver results for the 10 first seeds. Mean MIP gap is the gap between the solution and the inferior bound found.

also interesting results. Good results for global strategies can be explained by the resource-oriented approach of our algorithm (except for overloading and overloaded tenants, it considers only resource removal for tenant displacement). The very good performance of individual maximum constant piecewise, especially for large number of tenants shows that this fitter is very efficient, but needs a minimum number of tenants to become more efficient.

We have shown that our heuristic is fast, even when the number of tenants grows, and permits substantial savings for the BPMaaS providers. A gain of 20 % on the adapted strategy for 100 tenants corresponds to a mean of 1373.48 $ for two days. Moreover, it is possible to test multiple strategies, as the computing time stays low (for 200 tenants an individual strategy lasts a mean of 11.75 seconds, and 5.02 seconds for a grouped one). The longer duration for the individual strategy can be explained by the mechanics of the heuristic. Indeed, in this case we segment every tenant instead of all at once in the grouped strategy. Individual strategies could be more interesting to use in production environment. Indeed, even if we have not considered this constraint, migrating all the tenants together could have some side effects on QoS because all the data of the customers will migrate simultaneously, having negative effects on the available network bandwidth. Of course an individual strategy could give the same results if the tenants have identical workload patterns.

6 Conclusion

In this paper, we have proposed a new linear model for resource allocation and scheduling of BPM execution in the cloud, and a quick, simple and straightforward heuristic giving good results compared to naive approaches and solving of the model. This model relies on assumptions that makes it applicable in an operational setting. First, we consider customers as a whole (tenants) and not a distribution process instance by process instance. This reduces the scope of the calculation and avoids security issues regarding access to the business data. Second we assume that we can migrate a BPMS deployment from one installation to another in a reasonable time. This is not available in current systems but it is possible with very little service interruption.

We have tested it with data from an existing BPMS with a task metric. Of course, we could also use it with other metrics such as the number of processes and even for other services than BPMS. For instance, we could consider web servers and the throughput of HTTP queries as soon as they have a strong data management component. We can also use the algorithm with different temporal dimensions, hours as in the experimentation, but also minutes or seconds. We also plan to use it in an online manner, coupled with a predictive component that computes dynamically the expected load for the following time slots. This is our next step. We also think that can still improve our results using metaheuristics. Even if they are much better than with a naive approach, there is room for improvement as shown by our experiments with a solver for 5 tenants. Last, we want to test this heuristic with customer data, where the results should be better considering the load patterns we can identify (day/night cycle, working hours, lunch time, etc.).

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). The data and the results are available at : http://doi.org/10.5281/zenodo.401374 *. The source code of the framework is not free for now, except for the segmentation library, available at* https://github.com/guillaumerosinosky/Segmentation/.

References

1. Le, T.M.H., Alfredo, L.A., Choi, H.R., Cho, M.J., Kim, C.S.: A Study on BPaaS with TCO Model, IEEE (December 2014) 249–256 00003.
2. Rosinosky, G., Youcef, S., Charoy, F.: An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing Environment, IEEE (June 2016) 311–318 00001.
3. Schulte, S., Janiesch, C., Venugopal, S., Weber, I., Hoenisch, P.: Elastic Business Process Management: State of the art and open challenges for BPM in the cloud. Future Generation Computer Systems (2014) 00011.
4. Hoenisch, P., Schuller, D., Schulte, S., Hochreiner, C., Dustdar, S.: Optimization of Complex Elastic Processes. IEEE Transactions on Services Computing **9**(5) (September 2016) 700–713 00008.

5. Hoenisch, P., Schulte, S., Dustdar, S., Venugopal, S.: Self-Adaptive Resource Allocation for Elastic Process Execution, IEEE (June 2013) 220–227 00014.
6. Janiesch, C., Weber, I., Kuhlenkamp, J., Menzel, M.: Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure, IEEE (January 2014) 3818–3826 00007.
7. Euting, S., Janiesch, C., Fischer, R., Tai, S., Weber, I.: Scalable Business Process Execution in the Cloud. In: 2014 IEEE International Conference on Cloud Engineering (IC2E). (March 2014) 175–184 00006.
8. Hachicha, E., Assy, N., Gaaloul, W., Mendling, J.: A configurable resource allocation for multi-tenant process development in the cloud. In: International Conference on Advanced Information Systems Engineering, Springer (2016) 558–574 00005.
9. Sellami, W., Kacem, H.H., Kacem, A.H.: Elastic Multi-tenant Business Process Based Service Pattern in Cloud Computing, IEEE (December 2014) 154–161 00002.
10. Wolke, A., Tsend-Ayush, B., Pfeiffer, C., Bichler, M.: More than bin packing: Dynamic resource allocation strategies in cloud data centers. Information Systems **52** (August 2015) 83–95 00003.
11. Li, Y., Tang, X., Cai, W.: On dynamic bin packing for resource allocation in the cloud, ACM Press (2014) 2–11 00011.
12. Jaśkowski, W., Szubert, M., Gawron, P.: A hybrid MIP-based large neighborhood search heuristic for solving the machine reassignment problem. Annals of Operations Research (January 2015) 00002.
13. Brandt, F., Speck, J., Völker, M.: Constraint-based large neighborhood search for machine reassignment: A solution approach to the ROADEF/EURO challenge 2012. Annals of Operations Research (December 2014) 00000.
14. Das, S., Agrawal, D., El Abbadi, A.: ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud. ACM Transactions on Database Systems **38**(1) (April 2013) 1–45 00095.
15. Barker, S.K., Chi, Y., Hacigümüs, H., Shenoy, P.J., Cecchet, E.: ShuttleDB: Database-Aware Elasticity in the Cloud. In: 11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20, 2014. (2014) 33–43 00007.
16. Kang, J., Park, S.: Algorithms for the variable sized bin packing problem. European Journal of Operational Research **147**(2) (2003) 365–372 00116.
17. Lovrić, M., Milanović, M., Stamenković, M.: Algoritmic methods for segmentation of time series: An overview. Journal of Contemporary Economic and Business Issues **1**(1) (2014) 31–53 00006.
18. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, IEEE (2001) 289–296 00859.
19. Rosinosky, G., Youcef, S., Charoy, F.: A Framework for BPMS Performance and Cost Evaluation on the Cloud, IEEE (December 2016) 653–658 00000.
20. Sandrock, C.: Identification and Generation of Realistic Input Sequences for Stochastic Simulation with Markov Processes. In Cakaj, S., ed.: Modeling Simulation and Optimization - Tolerance and Optimal Control. InTech (April 2010) 00000 DOI: 10.5772/9035.
21. Optimization, G.: Gurobi Optimizer Reference Manual (2015)