An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing Environment

Guillaume Rosinosky, Samir Youcef, François Charoy

An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing environment

Guillaume Rosinosky
Bonitasoft
Grenoble, France
Email: guillaume.rosinosky@bonitasoft.com

Samir Youcef, François Charoy
Université de Lorraine, Inria, LORIA
Nancy, France
Email: samir.youcef@loria.fr, francois.charoy@loria.fr

***Abstract*—Even though the proven benefits of cloud computing paradigm, it must face a serious problem that can compromise its commercial success. It concerns the lack of efficient approach for using optimally the available resources. For this, several approaches have been proposed. However, they suffer from several shortcomings. For instance, often only one objective is taken into account expressing all operations in terms of cost. Furthermore, business processes should be insured with elasticity and multitenancy mechanism while adjusting the available resources to the dynamic load distribution. The proposed approach aims to optimize two conflicting objectives, namely the number of migrated tenants and the cost incurred using a set of resources. It allows to take into account the multi-tenancy property and the Cloud computing elasticity, and is efficient as shown by an extensive experimentation based on real data from Bonita BPM customers.**

I. Introduction

For years, software was sold and installed in customers premises and operated by their IT teams. This business model requires that the software is deployed on the user computer infrastructure who will operate it. The customer will ensure its maintenance and transition between versions. Recently, several businesses adopted a new paradigm where the software is operated in a data center and accessed remotely. This new way to consume software transfers the burden of IT management to the software provider. The later must ensure the required quality of service (QoS) at the lowest possible operating cost for its customers. Achieving that efficiently remains an issue. Here, we consider the delivery of a business process execution as a service as part of a Business Process Management System (BPMS) [1]. The goal of a BPMS is, among others, to control the execution of business processes, according to their model. Providing business process execution as a service (BPMaaS) means to accommodate the service requirement of a set of customers and to support the execution of their processes with the required level of quality of service, for the best cost. We want that at each point in time the available resources match the current demand as closely as possible. Thus, we must set up an elastic infrastructure [2] adapted to the specific requirements of business process execution. Generic auto scaling techniques provided by cloud providers do not usually take into account software usage metrics, but rather system or OS level metrics such as CPU or memory. We argue that we can use business processes and their history of usage to anticipate resource allocation need, and that it will be useful to achieve an efficient infrastructure elasticity for business process execution.

In this paper, we consider a service provider that wants to provide process execution as a service. The BPMS supports multi-tenancy, i.e. it can accommodate several customers on the same installation. It is possible to migrate a tenant from a BPMS installation to another BPMS installation with a limited interruption of service. Considering these assumptions, our goal is to propose a way to ensure the required quality of service for each tenant at the minimal cost while minimizing tenant migrations. That requires to adjust the number of computers to the load required for each tenant while minimizing the number of migrations of tenant from one set of resources to another. We propose an efficient bi criteria approach based on migrations number and cost optimization, solving iteratively for each number of migrations a repacking step for the existing resources, followed by a variable cost and size bin packing, and by a step of consolidation. We compare this approach to the solving of the corresponding model, using data based on customer information.

The reminder of the paper is organized as follows. In the next section, we discuss the problem in more details and we describe our hypothesis and the limits of this study. Section III summaries the existing work. Section IV gives our problem formulation of resource allocation to execute business processes in cloud computing environment. This section also presents the optimization objectives, namely the cost incurred using a set of resources and the number of migration. Section V presents our proposition to provide an approximate solution to the problem that can be computed in polynomial time. In section VI we describe the evaluation that we have conducted using Bonita BPM as an example of BPMS and AWS as the cloud provider. We show that it diverges very reasonably from an exact solution that could not scale. Section VII concludes the paper and describes its future extensions.

II. Motivation and hypothesis

Providing business process execution as a service requires for an operator to be able to manage execution of thousands of processes from hundreds of customers at the same time on its infrastructure. This requires an environment able to adjust itself to the load of the different customers automatically. We could rely on basic scalability techniques available in the cloud, adding more computing power in a BPMS cluster when needed. This is not so easy since a BPMS is database write intensive and we would certainly reach a maximum number of customer on the same cluster. We consider a more light weight approach based on a multi-tenant BPM solution where each tenant share the resources. We want to support as many tenants as possible at the minimal cost, while ensuring a defined quality of service. For that we assume that tenants can migrate from a BPMS installation to another with an acceptable service interruption. Our approach favor the management of hundreds of small BPMS installations over large scale cluster management. Also, we consider the distribution tenant by tenant, and not process by process as it can be done in similar research.

In the literature, many consider business process oriented optimization, at various level of elasticity completion, as we will see in the existing work part. In this paper, we consider a tenant-centric BPMaaS elasticity, by looking on tenant related usage data, as the total number of tenant BPM tasks. This avoids to replicate tenant data between installations and to maintain consistency among them.

Here we discuss about distribution of tenants and their activities on a cloud infrastructure. Our contribution is a new bi-objective model, an efficient tenant distribution method and algorithm which optimize operating cost for the service provider while ensuring limited number of interruptions. We only consider the change in the distribution from one hour to the other based on the knowledge of the load evolution of each tenant.

As many other applications, a BPM stack requires many software elements (see figure 1). It needs one or several instances of ACID compliant relational database systems which it uses to store process data, one or several web application servers who will contain the BPM Engine, load balancers in the case of clustered applications, and other services used in distributed applications.

We propose to measure BPM task execution throughput to estimate the required performance for tenants and the capability of various cloud configurations. Task execution throughput is strongly related to the transaction performances of the underlying database. In order to execute the tasks of a BPM process, the BPM engine may execute one or several transactions in its database, before, during and after the execution of the task. A transition in BPM process also triggers state changes in the database. Database transactions as a performance metric are often used in precedent works

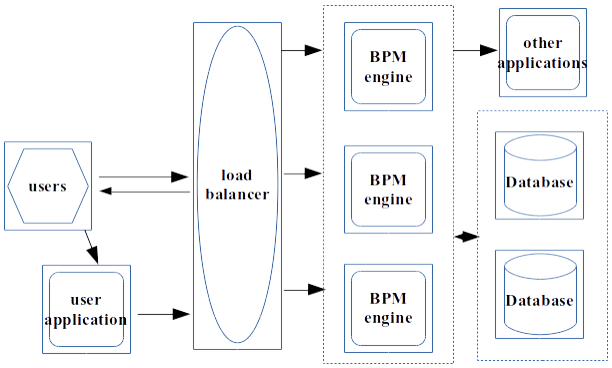

Figure 1. Simple BPMS architecture. Here we have a clustered version of a BPMS using a clustered database, behind a load balancer. The user can use its own application interfaced to the BPMS, or access directly to this last one via end-user interfaces. The BPMS may contact other applications via web services.

and in industry [3], [4], [5].

We assume that we have access to a public cloud provider with unlimited resources, and on demand billing. Compute resources are paid per hour and their price is constant. We consider that compute resources have stable performance. With the cloud resources, it is possible to build stacks of different sizes i.e. able to support different loads as defined before. For instance a stack can be composed of a compute instance for the database, another one for the BPMS. We name a defined group of compute instances a cloud configuration.

Each cloud configuration has a defined capability. We assume that each tenant can fit on at least one cloud configuration type, i.e the sum of its activities can be supported by the configuration type which can handle the most activity count per hour. Small tenant can share configurations. We neglect the effect of tenant execution on the other tenants executing on the same configuration : we will consider that tenants' workload adds up. For instance, a tenant with 10000 tasks per hour is similar to two tenants with 5000 tasks per hour.

In order to distribute the tenants on the cloud configuration, we need to have a way to estimate its required load. We have some knowledge on the behavioral patterns of every tenant for a given period of time. This information will be used to organize resources and distribute the load between configurations. This could be determined manually (in the contract part between the customer and the service provider), or using some prediction system.

Migration is the action of moving software and data from a cloud configuration to another. If a tenant needs more throughput than the current configuration can provide, either we migrate it to a bigger configuration in accordance with the QoS of the customer, or we migrate other tenants from the same configuration to free computing power. On the other way, we should move a tenant which requires

less resources. We assume that tenant awareness, automated migration, and tenant consolidation are totally supported by the BPM engine and its corresponding database systems.

However, migrations usually have negative effects on the origin and destination cloud configurations and the tenants they host. We consider that a tenant migration produces a service interruption that is acceptable for customers and that migration operations can be initiated and totally done in less than one hour, but we have chosen to take it into account in our approach. We can start building the configuration for hour $n+1$ at hour n. There are several solutions that are already available to do live migration of databases and servers very efficiently with minimal interruption such as [6].

These hypothesis provide us with a framework that remains realistic and that will allow us to define a model which can be optimized to ensure the best operation conditions for BPM tenants on a cloud. In the next section, we explore the current solutions for similar problems.

III. Related work

Schulte et al. [7] have made a review on the current status on BPM elasticity, and the different important criteria. Several attempts have been made recently on BPM elasticity, usually aiming to distribute processes without taking into account multi tenancy, and consider mainly CPU and RAM such as [8], [9], [10]. In [11], Sellami et al. propose a multi tenant approach based on customizable thresholds but it does not take into account migration cost or the database tier.

A lot of papers study virtual machine assignment in data-centers such as [12], [13]. However these works consider only the scheduling part, as the resources in data-center already exist.

Less attempts have been made on reassignment. A well known initiative is the Google Reassignment Problem [14] where the problem is mono objective and sums up load, balance, process, service, and program move cost. Several attempts to solve this problem have been made. The best one use variable neighborhood search [15], [16] , or constraint programming [17]. However these approaches aims at data centers, who have a predefined set of machines, and assume a fixed cost for migrations.

Other attempts such as multi-tenant relational database management system [5] or generic SaaS elasticity have been made for this problem, but there is no known work who take into account BPMS cloud migrations and cost in a multi-objective way.

IV. Our model for elastic multi-tenant BPM in the Cloud

The objective of our study is to minimize the total cost incurred using a set of resources and the number of migration that produce QoS degradation. Thus we face a multi-objective problem who can be tackled using three approaches: (i) the mono-criterion approach, (ii) the ϵ-constraints and (iii) the multi-criteria approach. The criteria that we consider are conflicting. If we consider only the cost objective part, an intuitive best solution could provoke the migration of all the tenants. But as the migrations can lead to QoS degradation, this is not worth considering. On the other side, minimizing the migration objective can lead to leave tenants on too expensive configurations. Therefore, the most appropriate approach is the multi-criteria one. We will consider the two objective functions simultaneously. The most used optimality notion when we deal with multi-criteria problems is the Pareto optimality concept defined as follows.

Definition 1. *We say that a solution $x \in X$ is a Pareto solution (or belong to the Pareto front) iff:*

- $\nexists x' \in X, \forall\ i \in \{1,...,n\}, f_i(x') \leq f_i(x)\ \wedge\ \exists\ j \in \{1,...,n\}, f_j(x') < f_j(x)$.

In order to formalize the problem, we assume that an installation of a BPMS is deployed on a configuration consisting in several cloud resources. We rely on several configuration types with different cloud instance types. We consider that a configuration type has a cost, and a throughput capability. This cost is the total of the cost of its composing cloud resources for a given time slot of one hour, since it is the unit of cost for most public cloud providers.

Let the following variables :

- $\mathcal{J}$ the set of configuration with m its cardinality
- C_j and W_j configuration j cost and capacity
- $\mathcal{I}$ the set of tenants with n its cardinality
- w_i the tenant i needed capacity
- ${x_j}^i(k)$ tenant i assigned to configuration instance j in time slot k
- $y_j(k)$ configuration j active in time slot k

We define an indicator function $\mathbb{1}_{\{{x_j}^i(k) \neq {x_j}^i(k+1)\}}$ which corresponds to actual tenant migration which will be equal to : $\begin{cases} 0 \text{ if } {x_j}^i(k) = {x_j}^i(k+1) \\ 1 \text{ if } {x_j}^i(k) \neq {x_j}^i(k+1) \end{cases}$

We aim to minimize the total configurations cost and the number of migrations for the time slot $k+1$ from the configurations distribution at time k where we apply the needed capacity at time $k+1$. The problem can be defined as follows:

$$\min f1 = \sum_{j}^{j \in \mathcal{J}} C_j y_j(k+1) \tag{1}$$

$$\min f2 = \sum_{j}^{j \in \mathcal{J}} \sum_{i}^{i \in \mathcal{I}} \mathbb{1}_{\{{x_j}^i(k) \neq {x_j}^i(k+1)\}} {x_j}^i(k+1) \tag{2}$$

We have the following constraints :

$$\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} {x_j}^i(k+1) = 1 \tag{3}$$

$$\forall j \in \mathcal{J} \sum_{i}^{i \in \mathcal{I}} w_i {x_j}^i (k+1) \leq W_j y_j (k+1) \tag{4}$$

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

Equation 1 presents the cost objective, and equation 2 the migration quantity objective. Equation 3 indicates that a tenant should be assigned to only one configuration. Equation 4 means that the sum of the required throughput of tenants on one resource should be less or equal than the capability of this resource.

For a defined number of resources, and without taking into account migrations, this is a classic assignment problem. However the resource allocation part and the migration part makes it more difficult to solve. Since, as we will precise in the next section, the problem is NP-hard, it is appropriate to propose heuristic algorithms rather than exact algorithms that would not scale. The following section describes the approach we propose.

V. An efficient approach for multi-tenant elastic business process execution in cloud context

The approach we propose is composed of two parts: (i) the resource allocation part and (ii) the scheduling part. Resource allocation and task scheduling are NP-hard problems. Greedy heuristics, integer and mixed integer linear programming, meta-heuristics, or constraint programming [18] are often used to deal with such problem. Greedy algorithms are heuristics who make the assumption that by aiming at the best local solution, a good global solution is reachable. This type of algorithm is usually simple and fast, but could get stuck in a local optimum. Integer linear programming (ILP) and mixed integer linear programming (MILP) are usual operational research algorithms, seeking to minimize an objective function under constraints with discrete variables (ILP), or mixed discrete and continuous variables. Solving these problems usually include LP relaxation in order to find valid solutions with continuous methods. However, depending on the problem size the computing can be long, and a solver is needed in order to resolve the problem.

Here, we choose to observe the migration number first since it is discrete, and to calculate the best cost for each migration number with an heuristic. Reducing the cost requires to reduce the number and the cost of configurations. In our approach, apart of the throughput constraint, we decided to not consider swapping tenants and to focus on resource reduction. We show an example of configurations and their tenants in figure 2.

The minimum number of tenants we have to move will be the number of tenants that can't fit anymore in their current resources. We divide them among two classes :

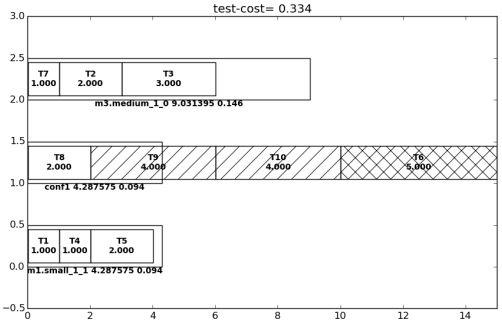

Figure 2. An example of distribution of tenants and configurations. Three configurations contain ten tenants in this situation. The outer boxes represent the configurations, and the inner ones the tenants. Crossed stripped boxes represent the unfit tenants, that can't stay anymore in their configuration. Single stripped boxes represent the overloading tenants. Blank boxes represent tenants that fit in their current resource.

- tenants that we must migrate because their future throughput is greater than the capability of their current resource. The migration of these tenants is mandatory. We name these type of tenants **unfit** tenants. These are the crossed line boxes in the figure 2 such as *T6*.
- tenant which, put together, are overloading their current resource. We choose to "remove" from the resources and force the migration of the biggest tenants in each overloaded resource until none is overloaded anymore. We name these type of tenants the **overloading** tenants. In figure 2 these are tenants *T9* and *T10*.

The main loop is described in algorithm 1. At each number of migrations, we consider the combination of resources containing precisely the number of tenants we wish to migrate if we add to it the number of mandatory tenants. This is a subset sum problem, where we have used a simple recursive approach. In figure 2, the minimum number of migrations is 3 (*T9*, *T10* and *T6*), and it is possible to also reassign 4 or 6 tenants - by removing respectively *conf1* or *m3_medium_1_0* configuration, but not 5 as there is no combination of resources hosting 5 tenants. However the number of possibilities increases exponentially, which will lead to memory and CPU time problems. For this, we must limit the number of resources combinations we wish to test. Moreover, there are cases where the algorithm could be slower than the exact method in particular when the resource quantity is high. Let's imagine we have 30 tenants, that are all hosted on their own configuration. In the case where we don't limit the number of subset sum combinations, for 15 migrations we will have C_{25}^{50} which means more than 10^{14} combinations to test. This number of migrations is very high and can't be computed in acceptable time. This is why it is important, even if it gives less interesting results, to limit the number of the subset sum combinations. Here we have considered 1000 resource combinations as a limit.

Algorithm 1 Main loop

```
1: procedure MAIN LOOP(tenant distribution for previous hour, needed throughput)
2:     resultNumberLimit = 1000
3:     unfit ← tenants whose throughput is bigger than their current resource
4:     overloading ← biggest tenants to remove until there is no more overloaded resource
5:     mandatoryTenants ← overloading ∪ unfit
6:     minMigrations = size(mandatoryTenants)
7:     maxMigrations = size(tenants)
8:     distribution = remove mandatoryTenants from distribution
9:     for i = minMigrations → maxMigrations do
10:        costByMigration[i] ← +∞
11:        possibleResourcesCombinations ← FINDSUBSETSUM(i, resources, result_numberlimit)
12:        for all resourceCombination ∈ possibleResourcesCombinations do
13:            newDistribution ← distribution - resources ∈ resourceCombination
14:            tenantsToRepack ← mandatoryTenants tenants ∈ resources ∈ resourceCombination
15:            REPACKING(newDistribution, tenantsToRepack)
16:            BINPACKING(newDistribution, tenantsToRepack)
17:            CONSOLIDATION(newDistribution, tenantsToRepack)
18:            if cost(newDistribution) ≤ costByMigration[i] then
19:                costByMigration[i] ← cost(newDistribution)
20:                distributionResult[i] ← newDistribution
21:    return distributionResult
```

For each possible combination, we then virtually suppress the concerned resources, and consider the **mandatory** tenants and the **orphan** tenants resulting from the suppression of the resources. In our heuristic we chose to replace at best these tenants in the existing resources - that we cannot delete without adding more migrations. For this part, we used a best fit decreasing approach, where we aim to load into the most appropriate resources the tenants.

After this step, we must create new resources for the remaining tenants. At this point, the existing resources can no longer be filled. We used for this part a variable cost bin packing heuristic with the remaining tenants, more precisely a variation of Iterative Best Fit Decreasing algorithm [19].

Since we have two separated parts, one looking to existing resources and the other creating new resources, both generate unused space that should be used at best. In order to solve this problem, we have added a consolidation step which tries to delete resources one by one and replace the concerned orphan tenants with the repacking procedure. This consolidation step is inspired by one of the operators of the local search used in [20]. We adapted it by looking on resources who contain only orphan tenants.

A configuration combination is kept only if is less costly than the previous computed, in order to obtain the Pareto front (as we compute from the minimum to the maximum number of migrations). Figure 3 shows an example of cost migration. The best solutions are for 3, and 7 migrations. Other solutions such as 6, 9, 10 migrations are useless as they involve more migrations for the same price. There is no solution under 3 migrations, and for 5 or 8 migrations.

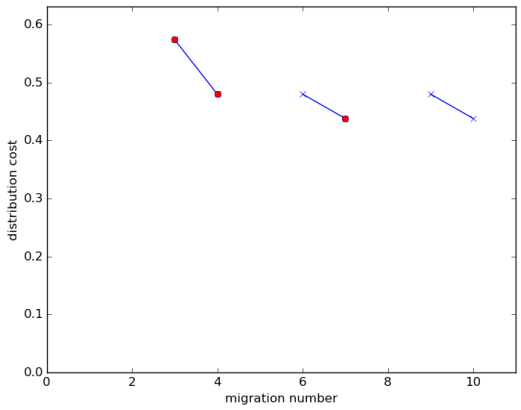

Figure 3. This represents the best results of our algorithm after we run it against the configuration in figure 2. The bigger dots represents the points in the Pareto frontier. Absence of dot means that there is no solution for the corresponding number of migrations.

VI. EXPERIMENTAL RESULTS

In this section, we present a summary of the results we obtained based on real data from Bonitasoft customers. In order to evaluate the quality of the results using our approach, given the particularity of the Pareto front due to the fact that the migration criterion is discrete, we compare for each migration number the cost between the heuristic and the results given by a solver.

We need to know first the size of each cloud configuration in term of throughput capability. We must also estimate the tenant size in term of BPM task throughput. For this, we use Bonita BPM [21] open-source business process management and workflow suite. Bonita BPM is

a multi-tenant BPMS where several customer can have their applications on the same Bonita BPM instance and uses a shared-schema strategy [22] in order to manage tenants. We have launched tests on Bonita BPM 7.0.3 using PostgreSQL 9.3 database, each on a separated EC2 instance on Amazon Web Services public cloud. A business process definition used by Bonitasoft for their internal performance tests has been used in order to compare the various configurations. The business process which we will name "standard process" contains twenty sequential automated tasks, each one launching one connector computing the 25 first Fibonacci numbers. We have launch each time 3000 processes with an injector tool on various cloud configurations and on different parallel process number injections. Our goal is to observe the correlation between the number of parallel processes and the average computing duration for each process considering each configuration, in order to find each configuration mean BPM task throughput.

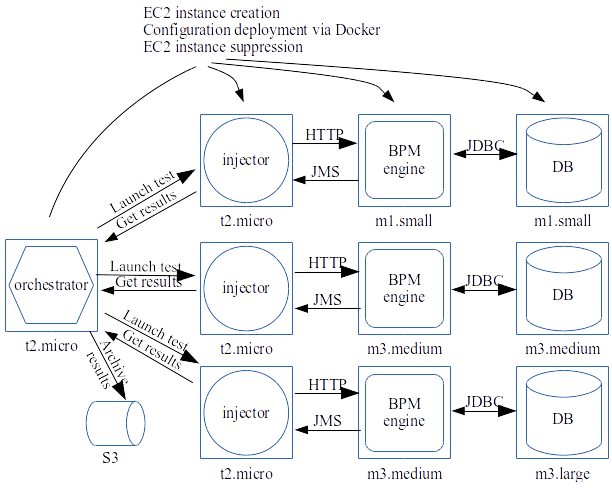

Figure 4. Test architecture used for the cloud configuration capability determination. Here we show three different databases and application servers.

As we don't know the duration of a task, we have computed the number of tasks for a given process duration. As the performances stay globally linear when we inject more processes in the engine (apart from the first ones, when we can assume there is not a lot of usage of the parallelism of the processor), we consider that it is possible to do a simple linear regression in order to find the capability in parallel processing. We have computed the corresponding process throughput for a duration of 10 second, by linear regression between the two corresponding values. We use this process throughput to compute the corresponding task throughput, by looking at the corresponding total duration and dividing it by the total number of tasks (here 60000). This gives us the mean task throughput we can expect for a mean duration of 10 second of standard process. In order to use realistic performances and prices, we have used these configuration's task throughput (in table I), and prices in our experimentation.

DB inst. type	AS inst. type	price	task TP	task TP per $
m1.small	m1.small	0.094	8.120	86.392
m3.medium	m3.medium	0.146	17.762	121.660
m3.medium	m3.large	0.219	24.669	112.644
m3.medium	m3.xlarge	0.366	25.293	69.107
m3.large	m3.xlarge	0.439	41.147	93.730
m3.large	m3.2xlarge	0.693	42.813	61.868
m3.xlarge	m3.2xlarge	0.839	45.274	53.962

Table I
PRICE, MEAN TASK THROUGHPUT, AND MEAN TASK THROUGHPUT BY DOLLAR FOR A MEAN STANDARD PROCESS DURATION OF 10 SECONDS

For the size of the tenants, we have used anonymous customer information fragments where we have observed the minimum and the maximum task throughput. In our experimentation, for each tenant i, we used an uniform distribution in interval $[w_i^{min}, w_i^{max}]$ to adjust the customer throughput, where w_i^{min} and w_i^{max} represent respectively the minimum and the maximum observed throughput. We have capped at the maximum configuration throughput the task throughput for each tenant in order to be able to fit them in cloud configurations. We show a summary of the data we collected in in table II.

customer	observed interval (in days)	minimum	maximum
A	4	2	45
B	1	14	16
C	45	0	45
D	7	1	3
E	45	5	45
F	550	0	4

Table II
SYNTHESIS OF THE USED CUSTOMER DATA. FOR EACH CUSTOMER, THE OBSERVED INTERVAL, THE MINIMUM AND THE MAXIMUM TASK THROUGHPUT PER SECOND FOR EACH HOUR.

Using a solver could suffice to have all required results. However for a high number of tenants it becomes very slow. This is why we propose an to use an heuristic.

In order to test our approach we have implemented this algorithm with a linear optimization solver. In order to simplify the multi-objective characteristic of the problem, we used the same migration number approach. For this we have transformed the objective function described in equation 2 in a constraint, as seen in equation 6, where M is the number of migrations for which we want to have the solution. Furthermore, $\mathcal{J}$ is initialized with all the possible tenants and configuration combinations.

$$\sum_{j}^{j\in\mathcal{J}} \sum_{i}^{i\in\mathcal{I}} \mathbb{1}_{\{x_j{}^i(k)\neq x_j{}^i(k+1)\}}\, x_j{}^i(k+1) = M \qquad (6)$$

This model can be easily resolved with linear programming solvers. We have used PuLP linear programming toolkit [23] coupled with Gurobi [24] linear solver on a

AWS c4.xlarge EC2 instance. A first step is to determine the minimum number of migrations, and compute the optimal cost without considering the migrations (by removing the constraint described in equation 6). We then compute the optimal cost for each number of migrations between the minimum number and stop computing once we reach the optimal cost. We have configured the solver with a time limit of 3 hours for a migration number computation.

For the heuristic test, we have launched 30 times the previously discussed uniform distribution for various number of tenants. We began with a distribution where all the tenants are on a minimum configuration, without considering the required load. We have then launched two times the heuristic. Indeed, this algorithm will be launched sequentially with an previously computed distribution. The results described in table III shows the second launch efficiency and duration. In order to compare the two methods, we launched the exact algorithm on the same tenant distributions. In some cases, the solver was not able to find a solution in the time limit.

Tenant quantity	Exact method results	Heuristic duration	Exact algorithm duration	Pareto front percent	Heuristic efficiency percent
5	30	0.003	0.128	85.55	1.41
10	30	0.047	0.864	85.38	0.97
20	30	5.243	13.625	81.55	1.97
30	29	78.56	579.31	78.56	2.26
40	29	1013.85	1850.66	66.22	4.10

Table III
HEURISTIC QUALITY VS. EXACT ALGORITHM (DURATION IN SECONDS, PARETO FRONT PERCENT AND EFFICIENCY)

The heuristic Pareto frontier percent corresponds to the ratio of migration optimum found in the heuristic which are optimum in the exact method. We compute the heuristic efficiency for all the heuristic Pareto frontier, with the distance to the corresponding price in the exact method. In addition, based on the results, we can see that the relative error does not exceed 4.1%. Furthermore, in the worst case that we studied, the Pareto front percentage is 66.22% and the heuristic stay faster than the exact algorithm duration.

The results we obtained show that this heuristic, while often taking a fraction of the time used in an optimization linear solver for linear programming software, gives good results with small errors, and most optimal migrations number obtained in the exact method. As the tenant quantity increases, less and less results are given in a acceptable running time by the exact algorithm. Even if the Pareto front percent decreases with time, the heuristic's cost efficiency stays at a very low level. The last point shows that the two Pareto front are very close.

We have seen that even if in this algorithm we consider only resources combinations and ignore tenants swaps between active configuration, the heuristic gives good results for most of the exact Pareto frontier, as the cost stay very close to the optimal cost.

Despite the use of an intuitive algorithm for the subset sum part by limiting the number of returned combinations, the obtained results are very encouraging. This algorithm should be able, without computing all the results, to give really a limited number of random solutions. In our case, using a approximate version of the subset sum method could greatly speed up our heuristic, as it consists in most of the running time from 40 tenants.

Another enhancement to speed up the heuristic is to elaborate a multi-threaded version of this algorithm in order to benefit from the architecture of multi-core processors while computing the different resources combinations. For instance, several resource combinations could be computed simultaneously.

VII. CONCLUSION

In this paper, we described an effective approach for business process execution as a service on the Cloud. This approach consider tenants as a whole and proposes to minimize two conflicting objectives, the execution cost and the migration number. We have validated this approach with an experimentation that shows the efficiency of our algorithm on data based on Bonitasoft customer usage, considering configuration cost based on AWS cost profiles, compared with an exact method. As [4] notes, distribution techniques are not totally orthogonal to consolidation methods, and some methods used for database, or in our case BPM tenants could be used with a higher level approach such as VM, or containers. With this heuristic architecture, it is conceivable to use multiple different algorithms for the bin packing part, the overloading tenants choice, or even the subset sum algorithm. For instance, testing multiple overloading tenants alternatives or using other variable size and cost bin packing algorithms such as [25] could be interesting and produce even better results. The next step for this work is to consider optimization for several consecutive time slots in order to optimize the cost not per hour but for a whole day, in order to take into account customer QoS requirements. We will also study the possibility to use other distributions that better adjust the throughput and customer distribution.

VIII. 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] M. Weske, *Business process management: concepts, languages, architectures*, 2nd ed. Heidelberg ; New York: Springer, 2012, 01764.

[2] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not." in *ICAC*, 2013, pp. 23–27. [Online]. Available: http://se2.informatik.uni-wuerzburg.de/pa/uploads/papers/paper-209.pdf

[3] J. Gao, P. Pattabhiraman, X. Bai, and W.-T. Tsai, "SaaS performance and scalability evaluation in clouds," in *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*. IEEE, 2011, pp. 61–71. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6139093

[4] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan, "Workload-aware database monitoring and consolidation," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 313–324. [Online]. Available: http://dl.acm.org/citation.cfm?id=1989357

[5] A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan, "Characterizing tenant behavior for placement and crisis mitigation in multitenant dbmss," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 517–528. [Online]. Available: http://dl.acm.org/citation.cfm?id=2465308

[6] 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*, vol. 9, p. 2010, 2010. [Online]. Available: https://128.111.41.26/research/tech_reports/reports/2010-09.pdf

[7] 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, 00011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X1400168X

[8] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal, "Self-Adaptive Resource Allocation for Elastic Process Execution." IEEE, Jun. 2013, pp. 220–227. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6676698

[9] C. Janiesch, I. Weber, J. Kuhlenkamp, and M. Menzel, "Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure." IEEE, Jan. 2014, pp. 3818–3826. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6759076

[10] 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.

[11] W. Sellami, H. H. Kacem, and A. H. Kacem, "Elastic Multi-tenant Business Process Based Service Pattern in Cloud Computing." IEEE, Dec. 2014, pp. 154–161. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7037661

[12] A. Wolke, B. Tsend-Ayush, C. Pfeiffer, and M. Bichler, "More than bin packing: Dynamic resource allocation strategies in cloud data centers," *Information Systems*, vol. 52, pp. 83–95, Aug. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0306437915000472

[13] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud." ACM Press, 2014, pp. 2–11. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2612669.2612675

[14] "Challenge ROADEF/EURO 2012 : Machine Reassignment." [Online]. Available: http://challenge.roadef.org/2012/en/sujet.php

[15] 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. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1571065312000297

[16] W. Jaśkowski, M. Szubert, and P. Gawron, "A hybrid MIP-based large neighborhood search heuristic for solving the machine reassignment problem," *Annals of Operations Research*, Jan. 2015. [Online]. Available: http://link.springer.com/10.1007/s10479-014-1780-6

[17] F. Brandt, J. Speck, and M. Völker, "Constraint-based large neighborhood search for machine reassignment: A solution approach to the ROADEF/EURO challenge 2012," *Annals of Operations Research*, Dec. 2014. [Online]. Available: http://link.springer.com/10.1007/s10479-014-1772-6

[18] M. Lombardi and M. Milano, "Optimal methods for resource allocation and scheduling: a cross-disciplinary survey," *Constraints*, vol. 17, no. 1, pp. 51–85, Jan. 2012. [Online]. Available: http://link.springer.com/10.1007/s10601-011-9115-6

[19] J. Kang and S. Park, "Algorithms for the variable sized bin packing problem," *European Journal of Operational Research*, vol. 147, no. 2, pp. 365–372, 2003, 00116. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221702002473

[20] V. Hemmelmayr, V. Schmid, and C. Blum, "Variable neighbourhood search for the variable sized bin packing problem," *Computers & Operations Research*, vol. 39, no. 5, pp. 1097–1108, May 2012, 00009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0305054811001961

[21] "Bonitasoft," 00000. [Online]. Available: http://www.bonitasoft.com/

[22] Frederick Chong, Gianpaolo Carraro, and Roger Wolter, "Multi-Tenant Data Architecture," Jun. 2006, 00199. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa479086.aspx

[23] S. Mitchell, M. OSullivan, and I. Dunning, "PuLP: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011, 00013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.4985&rep=rep1&type=pdf

[24] I. Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2015, 00000. [Online]. Available: http://www.gurobi.com

[25] M. Haouari and M. Serairi, "Relaxations and exact solution of the variable sized bin packing problem," *Computational Optimization and Applications*, vol. 48, no. 2, pp. 345–368, Mar. 2011. [Online]. Available: http://link.springer.com/10.1007/s10589-009-9276-z