This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

An efficient algorithm for the bursting of service-based applications in hybrid Clouds

Faouzi Ben Charrada and Samir Tata

Abstract—Enterprises are more and more using hybrid cloud environments to deploy and run applications. This consists in providing and managing software and hardware resources within the enterprise and getting additional resources provided externally by public clouds whenever this is needed. In this later case, deployment of new applications consists in choosing a placement of some components in the private cloud and some others in the public cloud. To tackle this NP-hard problem, we have proposed in a previous work an approximate approach based on communication and hosting costs induced by the deployment of components in the public cloud. In this paper, we go further and propose a new efficient algorithm adapted for service-based applications modelled that can be not only described as behavior-based but also as architecture-based compositions of services.

Index Terms—Cloud Computing, hybrid clouds, service-based applications, service bursting.

1 Introduction

Cloud Computing is a new delivery model for IT services based on Internet protocols. It typically involves provisioning of dynamically scalable and often virtualized resources at the infrastructure, platform and software levels. It addresses different fundamentals like virtualisation, scalability, interoperability, quality of service and failover mechanism [1].

Cloud environment differs from traditional environments on the fact that it (1) is massively scalable, (2) can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, (3) is driven by economies of scale, (4) can be dynamically configured (via virtualization or other approaches) and (5) can be delivered on demand [2].

Among other models, cloud environments can be public, private or hybrid. A public cloud (a.k.a. external cloud) is a cloud that provides cloud resources and services to the public. A private cloud (a.k.a. internal cloud) is an enterprise owned or leased cloud. In general, a hybrid cloud is a composition of two or more clouds of different models. Nevertheless, we define, in this paper, a hybrid cloud as a composition of one public cloud and one private cloud. Such a cloud is an environment in which an enterprise has its own private cloud that provides and manages some internal resources and only uses external resources provided by the public cloud when needed. This case especially occurs when the enterprise applications or platforms need to scale up and request additional resources that the private cloud is not able to provide. Getting new resources from a public cloud can handle this kind of situations.

In cloud environments, the diversity of requirements for hosting services in the Cloud makes the management of Cloud applications and resources a challenging task [3]. Particularly, scalability of applications and platforms, platform heterogeneity and application division across clouds can be difficult to manage [4]. In this paper, we are interested in application division across clouds and particularly the division of service-based applications (SBAs for short) in hybrid clouds.

SBAs consist in compositions of components providing business services. The composition can be architecture-based (*e.g.* described in Service Component Architecture specification [5] or UML component diagram [6]) or behavior-based (*e.g.* described in Business Process Execution Language specification [7] (BPEL) or Business Process Model and Notation [8] (BPMN)).

When deploying SBAs in a hybrid cloud, their division (a.k.a bursting) in the public and private clouds consists in choosing components/services to be deployed in the public cloud and those to be kept in the private cloud. Cloud bursting is used when there are spikes in workload in private or in local resources of an enterprise [9]. Several parameters would be taken into account to make the division decision. We can cite among others security, privacy as well as communication and hosting costs generated by the relocation of services to the public cloud. In this paper, we are mainly interested in communication and hosting costs and we do not deal with security, privacy and other related properties. While we believe that these issues are important, the communication and hosting costs criteria discussed here are complex

- *Faouzi Ben Charrada is with the Department Computer Science, Faculty of Science of Tunis, University of Tunis El Manar, Tunis, Tunisia.*
  E-mail: f.charrada@gnet.tn
- *Samir Tata is with Institut Mines-Telecom, Telecom SudParis, UMR CNRS Samovar, Evry, France*
  Email: Samir.Tata@mines-telecom.fr

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

enough in themselves to deserve separate treatment.

When we visited existing work on placement of resources at different levels such as Software as a Service (a.k.a. SaaS), Platform as a Service (a.k.a. PaaS), Infrastructure as a Service (a.k.a. IaaS), we found out that they are not suitable to the considered problem of this paper. Indeed, they do not consider communication between services or do not consider hybrid situations (see Section 2 for more detail).

As we will show, in Section 3, the formulation of the bursting of services within a hybrid cloud consists in minimizing a quadratic function of several variables subject to linear constraints on these variables. It is well known that this type of problem is NP-Hard [25]. So optimal solutions of such hard problem need the running of programs that have exponential time complexity. While in some situations, e.g. for batch tasks, this can be tolerated, it is not the case for deployment of applications in the Cloud. Otherwise, this task can be a bottleneck of Cloud management processes. Consequently the challenge here is to propose an algorithm that behaves in polynomial time and approximates, in an acceptable way, the optimal placement solution.

In a previous work [10], we have proposed an algorithm that behaves in a polynomial time and approximates the optimal bursting solution. The proposed algorithm is efficient when the SBA can be represented by a sparse graph of services. It is mainly the case when the composition of services of the considered SBA is behavior-based.

In this paper we propose a new and more efficient algorithm that produces solutions which are very close to the optimal ones. Our contribution is efficient not only for the bursting of behavior-based compositions but also for architecture-based compositions of services.

The rest of this paper is organized as follows. Section 2 presents the state of the art related to the placement and bursting issues in the cloud. In Section 3, we represent SBAs as graphs and formulate the problem of bursting of SBAs in hybrid Clouds. Our proposed efficient algorithm for the bursting of SBAs in hybrid clouds is presented in Section 4. Our experiments are detailed in Section 5. Section 6 concludes the paper and gives our future work.

2 State of the art

The problem of placement of resources (virtual machines, web servers, software components and data) in cloud environments has been tackled from different point of views (IaaS, PaaS and/or SaaS), while considering different types of clouds (private/public, hybrid and federation of clouds) and different types of criteria (hosting, communication, QoS, etc.). In the following, we present approaches for the placement of resources in private/public clouds. After that, we present approaches for the bursting of resources in hybrid clouds (including federation of clouds).

2.1 Placement of resources in private/public clouds

We present below some approaches for placement at the IaaS level [11], at the PaaS level [12] and at the SaaS level [13], [14].

In [11], the authors present algorithms to allocate platform resources for service placement. These algorithms take into account communication resources, hosting resources and QoS. The considered resources are mainly instances of services that should be deployed on top of web servers.

In [12], authors present an approach for optimizing the placement of virtual resources at the level of infrastructure across multiple clouds. They developed a broker that allows the deployment and management of virtual resources based on the placement results.

In [13], the authors present a penalty-based genetic algorithm for the placement of components and data (used by these components) that optimize the SaaS performance.

Authors of [14] consider a placement problem for SaaS components in a multi-tenant architecture. Components are placed in a set of available platform nodes while optimizing the resource usage in each node.

All these approaches are suitable for the placement in public or private clouds but not for hybrid clouds. Indeed, in hybrid cloud communication costs between services depend on the placement of services where public and private communications should be differentiated. Indeed, the placement of applications in public/private clouds is a matter of placement of a virtual network (that represents an application) on a physical network that represents the cloud nodes and links. The problem is then to minimize the communication (and other non functional properties) in the placement. The communication is of the same type (in a cloud) while in our problem and context the communications are of two types (hybrid communication and public communication). Then, in our case the problem, its formulation and its cost function are different from those of the approaches referenced above. This case deserves a separate treatment.

2.2 Bursting of resources in hybrid clouds

In addition of the abovementioned approaches, we present approaches that optimize bursting of resources in hybrid clouds.

A scheduling model for optimizing virtual cluster placements across available cloud offers was proposed in [15] and in [16]. Placements are based on different criteria such as optimization (*e.g.* average prices), user constraints (*e.g.* performance) and environmental conditions (*e.g.* static vs. dynamic). This

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

approach is not applicable for our context since placements do not consider communication cots.

In [17], the authors present an approach that tries to minimize the cost of provisioning workload in hybrid IaaS clouds which consists in outsourcing partial workloads from private to public clouds. The presented work focuses on deadline-constrained workloads, workloads that cannot be migrated, memory and CPU. These criteria are considered in a binary integer programming problem formulation. The authors argued that when scheduling applications in public clouds, their approach behaves well but they recognized that in the hybrid setting, their approach's performance decreases drastically.

In [18], [19], [20], the authors propose several approaches for placement of virtual machines across multiple clouds. While these approaches can be adapted for the placement in hybrid clouds they dont take into account relation between services (or virtual machines).

In [21], the authors present an approach mainly composed of two steps for partitioning software services in hybrid cloud. Fist, application to be partitioned is profiled to produce a dependency graph that represents connected functions of the application. This first step assume that the application in developed with the Java programming language. Second, an optimization algorithm, based on an integer programming approach, is applied to divide the dependency graph across public and private clouds. The computational complexity of this algorithm is not provided but the integer programming algorithms are in NP [22].

3 Deployment of applications in hybrid clouds

In this section we define service-based applications that we consider in this paper. We show how these applications are represented as graphs. We describe the problem of service bursting and formulate the optimal solution as a minimizing of a function of a quadratic programming problem.

3.1 Service-based application

SBAs consist in assembling of a set of services using appropriate service composition specifications that can be architecture-based like Service Component Architecture (SCA [5]) and UML component diagram [6] or behavior-based like Business Process Execution Language (BPEL [7]) and Business Process Model and Notation (BPMN [8]). In the following we define these two types of compositions and provide illustrative examples. Then we model SBAs and deployed SBAs as graphs. After that, we state the placement problem and give its formulation.

3.1.1 Behavior-based compositions

A SBA composed using a behavior-based specification can be described as a structured process which consists of a set of process nodes and transitions between them. A process node can be service, Or-Join, Or-Split, And-Split or And-Join. And-Joins and Or-Joins should have at least two ingoing transitions and at least one outgoing transition. And-Splits and Or-Splits should have at least one outgoing transition and at least two ingoing transitions. Non-initial and non-final services have one ingoing transition and one outgoing transition.

Definition 1 (Structured process): A structured process is inductively defined in [23] as follows:

- A process consisting of a single service is a structured process. This service is both initial and final.
- Let X and Y be structured processes. The concatenation of these processes where the final node of X has a transition to the initial node of Y, then also is a structured process. The initial node of this process is the initial node of X and its final node is the final node of Y.
- Let $X_1, ...X_n$ be structured processes and let j be an Or-Join and s an Or-Split. The process with as initial node s and final node j and transitions between s and the initial node of X_i ($i \in \{1, 2, ..., n\}$) and between the final node of X_i ($i \in \{1, 2, ..., n\}$) and j is then also a structured process. The initial node of this structured process is s and its final node is j.
- Let $X_1, ...X_n$ be structured processes and let j be an And-Join and s an And-Split. The process with as initial node s and final node j and transitions between s and the initial node of X_i ($i \in \{1, 2, ..., n\}$) and between the final nodes of X_i ($i \in \{1, 2, ..., n\}$) and j is then also a structured process. The initial node of this structured process is s and its final node is j.

As an example of a structured process, we consider a business process of an online shopping purchase order of a clothing store called ClothStore (fictitious name). ClothStore offers products to its customers, interacts with two suppliers and a shipper for processing orders. It holds certain products in stock, and orders others from suppliers in case of product lack. The second supplier is contacted only if the first one hasn't the required quantity or articles.

The structured process of ClothStore is illustrated in the BPMN diagram shown in Figure 1. The customer sends a purchase order request with details about the required products and the needed quantity. Upon receipt of customer order, the seller checks product availability. If some of the products are not in stock, the alternative branch ordering from suppliers is executed. When all products are available, the choice of a shipper and the calculation of the initial price of the order are launched. Afterwards, the shipping price

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

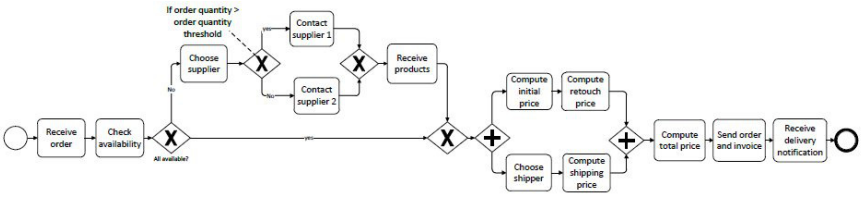


Fig. 1. Example of an SBA application modelled as a structured process

and the retouch price are computed simultaneously. The total price is then computed in order to send invoice and deliver the order. Finally, a notification is received from the shipper assuring that the order is already delivered.

3.1.2 Architecture-based compositions

A SBA composed using an architecture-based composition can be described as a set of linked components. A component provides one or more services. It may consume one or several references, which are services provided by other components. Connection of one reference and one service is realized by a wire.

As a running example, we consider the on-line store example illustrated in Figure 2 using a UML component diagram [6]. This diagram illustrates various components and services they offer to form a SBA. It shows how components reference services offered by other components. Offered services are defined through interfaces. These interfaces represent the contract between components.

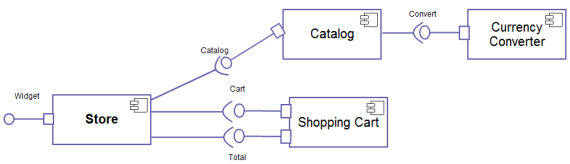


Fig. 2. Example of an on-line store (adapted from [24])

The example is a composition of four services. The Store service provides the interface of the on-line store. The Catalog service which the Store service can ask for catalog items provides the item prices. The CurrencyConverter service does the currency conversion for the Catalog service. The ShoppingCart service is used to include items chosen from the Catalog service.

In the following sections, we refer to this running in order to explain our concepts and motivate our work.

3.1.3 Services in SBAs

Services that compose SBAs can be identified by URIs and characterized by their interfaces, bindings and implementations. Services may be implemented with several programming languages (C++, Java, etc.), support several communication protocols (RMI, SOAP/HTTP, etc.) and/or run on several hosting frameworks (POJO, .NET framework, component-based platform, etc.). A service implementation requires specific platform resources for its deployment such as a container with specific sizing. A connection between two services is characterized by the amount of data flow to be communicated between services. For example, the Catalog service is bound using the JSONRPC binding, and the ShoppingCart service is bound using the ATOM binding. The Store service is implemented in HTML and JavaScript and the Catalog service is implemented in Java.

3.2 SBA and deployed SBA graphs

3.2.1 SBA graphs

Based on the above considerations, we can model a SBA like the one presented in Figure 1 as a graph. Services, Or-Join, Or-Split, And-Split and And-Join nodes will be represented by graph nodes and connections/transitions between services will be represented by edges. Nodes are identified by an ID (a *URI* or a *number*) and characterised by an amount of units of hosting resources. Edges are characterised by an amount of communications units which refers to the amount of traffic that is transferred on the considered edge.

Definition 2 (SBA graph): A SBA graph is a 5 tuple $\langle S, E, h, c, l \rangle$ that represent a SBA and its deployment in a hybrid cloud where:

- S is a set of services, Or-Join, Or-Split, And-Split and And-Join nodes composing the considered application (when the application is architecture-based S does not contain Or-Join, Or-Split, And-Split and And-Join nodes).
- $E \subseteq S \times S$ is the service connection set.
- $h : S \rightarrow \mathbb{R}^+$ is a hosting function that for each service associates a positive number that represents its hosting quantity of needed resources for its deployment. h can also be naturally extended to equivalent set of services as the sum of its application to all the nodes of the set. For the particular case of empty set, $h(\emptyset) = 0$.
- $c : E \rightarrow \mathbb{R}^+$ is a communication function that for each edge $e = \langle s_1, s_2 \rangle$ associates a positive number that represents the communication quantity between s_1 and s_2. The function c can also be naturally extended to equivalent set of edges as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

the sum of its application to all the nodes of the set. For the particular case of empty set, $c(\emptyset) = 0$.
- $l : S \rightarrow \{0, 1\}$ is a location function that for each service associates 0 if it is deployed in the private cloud and 1 if it is deployed in the public cloud.

Figure 3 represents the SBA graph of the SBA of Figure 2.

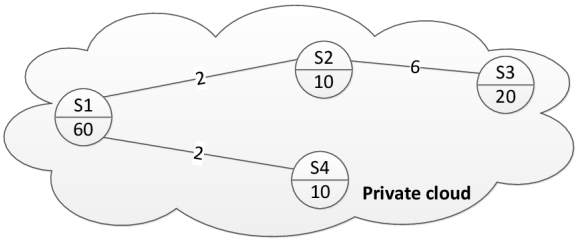


Fig. 3. The SBA graph of the on-line store

3.2.2 Deployment SBA on hybrid clouds

A deployed SBA in a hybrid cloud is represented by a graph where some services are deployed in the public cloud and the others are deployed in the private cloud.

Definition 3 (Deployed SBA graph): A deployed SBA graph is a 5 tuple $\langle G, Private, Public, icc, occ\rangle$ that represent a SBA and its deployment in a hybrid cloud where:

- $G = \langle S, E, h, c, l\rangle$ is a SBA graph as it is defined in Definition 2.
- $Public$ and $Private$ are a partition of S ($Public \subseteq S$, $Private \subseteq S$, $Public \cap Private = \emptyset$ and $Public \cup Private = S$).
- $icc : S \rightarrow \mathbb{R}^+$. icc stands for the inner communication cost. It is the function that for each service s_i in $Public$ (respectively $Private$) calculates the sum of the communication cost between s_i and the other services in $Public$ (respectively $Private$). Note that icc can be formulates with the function l and c.
- $occ : S \rightarrow \mathbb{R}^+$. occ stands for the outer communication cost. It is the function that for each service s_i in $Public$ (respectively $Private$) calculates the sum of the communication cost between s_i and the other services in $Private$ (respectively $Public$). Note that occ can be formulates with the function l and c.

Figure 4 represented the deployed SBA graph related to the SBA of Figure 2 where $Private = \{s_1\}$ and $Public = \{s_2, s_3, s_4\}$

When a component of a SBA is deployed in the public cloud, the provider of the application will pay for the hosting resources and communication units the application consumes. To model such costs, we define the following:

- α is the cost of a hosting unit of resource.
- β_1 is the cost of a communication unit between the public and the private cloud.
- β_2 is the cost of a communication unit inside the public cloud.

The cost related to the deployment of the applications of the running example as it is presented in Figure 4 for $\alpha = 5$, $\beta_1 = 10$ and $\beta_2 = 2$ is $5 \times (10 + 10 + 20) + 10 \times (2 + 2) + 2 \times (6) = 252$.

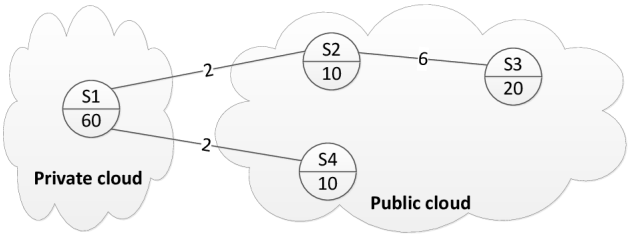


Fig. 4. The on-line store deployed on a hybrid cloud

3.3 Problem statement and its formulation

3.3.1 Problem statement

We are interested in this paper in deployment of SBAs in hybrid cloud that we consider as a composition of one private cloud and one public cloud. It is obvious that for an enterprise, applications should be deployed in its private cloud as long as the needed resources can be provided by the private cloud. But, when deployment in the public cloud is required? We have identified the following three cases:

- Already deployed applications request more resources the private cloud could not provide.
- Already deployed applications release resources so that a re-deployment can be envisaged to release allocated resources in the public cloud.
- New deployment requests to be fulfilled can not be satisfied by the private cloud.

In this paper, we are interested in deciding about services to be deployed in the public cloud based on some parameters we detailed below. The need of resources from the public cloud can be quantified in terms of amount of hosting units (units of platform resources required). This amount refers as hosting quantity and noted *HQ*. When it is necessary to use the public cloud, one has to decide about services of applications that should be chosen to be deployed in the public cloud. The quantity of required platform resources of the chosen services has to be greeter or equal to *HQ*.

In addition to their hosting cost, services that have to be deployed in the public cloud generate costs related to communications between them (communication inside the public cloud) and communication between them and services of applications deployed in the private cloud (communication between the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

private and the public cloud). Of course, these costs (hosting and communication) have to be supported by the enterprise.

Note that we do not consider here the communication and hosting cots within the private cloud. We are interested in minimizing the costs should be paid to the provider of the public cloud. In fact, we consider that almost all the private resources are consumed. They have to be supported in any case. So decide about services to be deployed in the public cloud is equivalent to minimize costs of hosting in the public cloud and communications between the private and the public cloud and in the public cloud. But this minimization problem is subject to a constraint. Indeed, the hosting quantity of deployed services in the public cloud is to be greeter or equal to HQ.

3.3.2 Problem formulation

The deployment of a SBA modelled as SBA graph as it is described above can be formulated as a programming problem as follows.

Minimize: $H + PC + HC$

Subject to:

$\sum_{i=1}^{n} h(s_i) \times l(s_i) \geq HQ$

Where:

$H = \sum_{i=1}^{n} \alpha \times h(s_i) \times l(s_i)$

$PC = \sum_{e=\langle s_i, s_j \rangle \in E} \beta_2 \times c(e) \times l(s_i) \times l(s_j)$

$HC = \sum_{i=1}^{n} \sum_{j \ s.t. \ e=\langle s_i, s_j \rangle \in E} \beta_1 \times c(e) \times l(s_i) \times (1 - l(s_j))$

H is the sum of hosting costs of services deployed in the public cloud. Indeed, the expression $\alpha \times h(s_i) \times l(s_i)$ (where α is the cost of a hosting unit of resource, $h(s_i)$ is hosting quantity of service s_i and $l(s_i)$ equals to 1 if s_i is in the public cloud and 0 otherwise) equals to the cost of the hosting of s_i if this service is deployed in the public cloud.

PC is the sum of public communications (communications between services deployed in the public cloud). Indeed there is a public communication between s_i and s_j if they are both deployed in the public cloud, *i.g.* $l(s_i) \times l(s_j)$ equals to 1.

HC is the sum of hybrid communications (communications between services deployed in the public cloud and those deployed in the private cloud). Ideed there is a hybrid communication between s_i and s_j if one of them is deployed in the public cloud and the other one is deployed in the private cloud, *i.g.* $l(s_i) \times l(1-(s_j))$ equals to 1 or $l(s_j) \times l(1-(s_i))$ equals to 1.

4 Algorithms for an efficient bursting

In this section, we propose two algorithms for the bursting of service-based applications in hybrid clouds. These algorithms are based on three procedures called $Forward$, $Backward$ and $Refinement$. In the following we present these procedures (Section 4.1). We present after the first algorithm based on the $Forward$ and $Backward$ procedures [10] that is suitable for sparse graphs which is the case of behavior-based compositions of services (Section 4.2).

The second algorithm we propose here is based on three combinations of the $Forward$, $Backward$ and $Refinement$ procedures. It is an efficient algorithm that chooses the cheapest solution of these different combinations (Section 4.3).

4.1 $Forward$, $Backward$ and $Refinement$ procedures

The $Forward$ procedure consists in choosing services from the private cloud ($Private$ set) to be moved to the public cloud ($Public$ set). The $Backward$ procedure tries to tune the result of the first step by moving services from public cloud back to the private cloud. The $Refinement$ procedure tries to tune the result for the second step by moving services again from the private to public cloud.

The $Forward$ procedure (see Algorithm 1) is to be executed while the sum of hosting quantities of moved services does not exceed HQ (see Algorithm 1, line 3). This step is based on the function $cost(s_i)$ that calculates, for each service s_i in the private cloud, the cost caused by its bursting to the public cloud. It takes into account inter-cloud communication, intra-public cloud communication and hosting in the public cloud costs (see lines 5-6). The service to be chosen is the one with the minimum cost caused (see lines 7-10 and 12-13).

The $Backward$ procedure (see Algorithm 2) tries to move back services already in the $Public$ set to the $Private$ set while (1) reducing global cost and (2) enforcing that the hosting values of services in $Public$ set are greater or equal to HQ. In fact, line 5 of Algorithm 2 ensures this later condition whereas line 6 calculates the price to gain or lose of moving back services from the public to the private cloud. If the minimum cost (lines 7-10) is negative (price to win), the service for which the cost is minimum is removed from $Public$ and added back to $Private$ (lines 12-14). In this case, $true$ is assigned to the variable $MoveBack$ (line 15). This value allows for another iteration to look for an additional service to moved back to $Private$ set (line 2).

The $Refinement$ procedure tries to move (again) services from the $Private$ to the $Public$ set while reducing global cost. In fact line 6 of Algorithm 3 calculates the price to gain or lose of moving services

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

Algorithm 1: Forward procedure

Require: $\langle S, E, , h, c, l\rangle$: SBA graph
Ensure: $Public$: set of application nodes in the public cloud
Ensure: $Private$: set of application nodes in the private cloud
1: $Private \leftarrow S$;
2: $Public \leftarrow \emptyset$;
3: **while** $h(Public) < HQ$ **do**
4: $mincost \leftarrow +\infty$;
5: **for all** $s_i \in Private$ where s_i is a service **do**
6: $cost(s_i) \leftarrow$ $\beta_1 \times icc(s_i) + (\beta_2 - \beta_1) \times occ(s_i) + \alpha \times h(s_i)$;
7: **if** $cost(s_i) < mincost$ **then**
8: $mincost \leftarrow cost(s_i)$;
9: $s_{min} \leftarrow s_i$
10: **end if**
11: **end for**
12: $Private \leftarrow Private \setminus \{s_{min}\}$;
13: $Public \leftarrow Public \cup \{s_{min}\}$;
14: **end while**

Algorithm 2: Backward procedure

Require: $\langle S, E, , h, c, l\rangle$: SBA graph
Require: $Public$: set of application nodes in the public cloud
Require: $Private$: set of application nodes in the private cloud
Ensure: $Public$: set of application nodes in the public cloud
Ensure: $Private$: set of application nodes in the private cloud
1: $MoveBack \leftarrow true$;
2: **while** $MoveBack$ **do**
3: $mincost \leftarrow +\infty$;
4: $MoveBack \leftarrow false$;
5: **for all** $s_i \in Public$ *where* $h(Public) - h(s_i) \geq HQ$ **do**
6: $cost(s_i) \leftarrow$ $(\beta_1 - \beta_2) \times icc(s_i) - \beta_1 \times occ(s_i) - \alpha \times h(s_i)$;
7: **if** $cost(s_i) < mincost$ **then**
8: $mincost \leftarrow cost(s_i)$;
9: $s_{min} \leftarrow s_i$
10: **end if**
11: **end for**
12: **if** $mincost < 0$ **then**
13: $Private \leftarrow Private \cup \{s_{min}\}$;
14: $Public \leftarrow Public \setminus \{s_{min}\}$;
15: $MoveBack \leftarrow true$;
16: **end if**
17: **end while**

from the private to the public cloud. If the minimum cost (lines 7-10) is negative (price to win), the service for which the cost is minimum is removed from $Private$ and added to $Public$ (lines 12-14). In this case, $true$ is assigned to the variable $MoveBack$ (line 15). This value allows for another iteration to look for an additional service to moved back to $Private$ set (line 2).

Algorithm 3: Refinement procedure

Require: $\langle S, E, , h, c, l\rangle$: SBA graph
Require: $Public$: set of application nodes in the public cloud
Require: $Private$: set of application nodes in the private cloud
Ensure: $Public$: set of application nodes in the public cloud
Ensure: $Private$: set of application nodes in the private cloud
1: $Move \leftarrow true$;
2: **while** $Move$ **do**
3: $mincost \leftarrow +\infty$;
4: $Move \leftarrow false$;
5: **for all** $s_i \in Private$ **do**
6: $cost(s_i) \leftarrow$ $\beta_1 \times icc(s_i) + (\beta_2 - \beta_1) \times occ(s_i) + \alpha \times h(s_i)$;
7: **if** $cost(s_i) < mincost$ **then**
8: $mincost \leftarrow cost(s_i)$;
9: $s_{min} \leftarrow s_i$
10: **end if**
11: **end for**
12: **if** $mincost < 0$ **then**
13: $Private \leftarrow Private \setminus \{s_{min}\}$;
14: $Public \leftarrow Public \cup \{s_{min}\}$;
15: $Move \leftarrow true$;
16: **end if**
17: **end while**

4.2 Forward-Backward (FB) algorithm

The $Forward - Backward$ algorithm (FB algorithm for short) is a first approximate bursting algorithm that calls first the $Forward$ procedure and then the $Backward$ procedure (see Algorithm 4) [10].

Algorithm 4: FB procedure

Require: $Graph : \langle S, E, , h, c, l\rangle$: SBA graph
Ensure: $Public$: set of application nodes in the public cloud
Ensure: $Private$: set of application nodes in the private cloud
1: $Public$, $Private \leftarrow Forward(Graph)$
2: $Public$, $Private \leftarrow Backward(Graph, Public, Private)$

4.3 Forward-Backward-Refinement (FBR) algorithm

After the presentation of the procedures $Forward$, $Backward$ and $Refinement$ and the first algorithm called $Forward - Backward$, Algorithm 5 presents our efficient approach called $Forward - Backward - Refinement$ algorithm (FBR algorithm for short). It is based on three combinations of the procedures $Forward$, $Backward$ and $Refinement$. The first combination calls first the $Forward$ procedure, then the $Backward$ procedure and finally the $Refinement$ procedure (see Algorithm 5, lines 1-3). The second combination calls first the $Forward$ procedure, then the $Refinement$ procedure and finally the $Backward$ procedure (see Algorithm 5, line 1 and lines 4-5). The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

third combination places all the application nodes in the public cloud then it calls the $Backward$ procedure and finally calls the $Refinement$ procedure (see Algorithm 5, lines 6-9). Finally, our algorithm chooses the cheapest solution of these above presented combinations (see Algorithm 5, lines 10-12).

Algorithm 5: FBR algorithm

Require: $G : \langle S, E, , h, c, l\rangle$: SBA graph
Ensure: $Public$: set of application nodes in the public cloud
Ensure: $Private$: set of application nodes in the private cloud
1: $Public$, $Private \leftarrow Forward(G)$
2: $Public_1$, $Private_1 \leftarrow Backward(G, Public, Private)$
3: $Public_1$, $Private_1 \leftarrow Refinement(G, Public_1, Private_1)$
4: $Public_2$, $Private_2 \leftarrow Refinement(G, Public, Private)$
5: $Public_2$, $Private_2 \leftarrow Backward(G, Public_2, Private_2)$
6: $Private_3 \leftarrow \emptyset$
7: $Public_3 \leftarrow Graph.S$
8: $Public_3$, $Private_3 \leftarrow Backward(G, Public_3, Private_3)$
9: $Public_3$, $Private_3 \leftarrow Refinement(G, Public_3, Private_3)$
10: choose i s.t. $cost(Public_i, Private_i) \leq cost(Public_j, Private_j), j = 1, 2, 3$
11: $Public \leftarrow Public_i$
12: $Private \leftarrow Private_i$

In terms of complexity, it is obvious that Algorithms 5 and 4 are polynomial whereas the initial problem of bursting we define in Section 3.3.2 is NP-Hard. Now, to evaluate the quality of Algorithm 4 and Algorithm 5 regarding the closeness of the provided solutions to the optimal ones, we show in Section 5 experiments we have conducted.

5 Experiments

In service research field, there are two types of compositions of services: behavior-based and architecture-based compositions of services. Behavior-based compositions of services are generally sparse graphs where nodes represent services and operators and links represent dependencies between services and operators. Architecture-based compositions of services can be sparse or dense graphs where nodes represent services and links represent dependencies between services.

To evaluate our proposed algorithms for the placement of services in hybrid clouds, we compared its quality of responses against an algorithm that calculates for each graph all possible partitions ($Public$, $Private$) and evaluate them to choose an optimal solution that satisfies the quadratic programming problem as it is formulated in Section 3.3.2.

We considered in all our experiments that the cost of a communication unit between the public and the private clouds is greater than the cost of a communication unit inside the public Cloud (*i.e.* $\beta_1 = 1$ and $\beta_2 = 10$). In addition, we have varied in our experiments the cost of a hosting unit ($\alpha \in \{6, 8, 12, 15, 20, 25\}$). α is sometimes greater than β_1 and some times β_1 is greater than α. This is to ensure experiments that cover different situations of α compared to β_1 and vice versa. As for HQ, it is varied from 5% to 95% of the total hosting of each graph and takes 18 different values. The distance between the optimal and our approximate solutions can be qualified by the percentage of loss that is equal to $(CAS\text{ - }COS)\times 100/COS$, where COS is the cost of an optimal solution and CAS is the cost of the approximate solution.

5.1 Experiments on architecture-based compositions

To evaluate our Algorithms, we have treated graphs with a number of nodes between 11 and 18. We have considered 10 randomly generated SBA graphs which can reflect architecture-based compositions of services. Some of these graphs are dense and some others are sparse. The density of graphs is expressed in terms of percentage computed as the 100 times of the ratio of number of edges out of the number of all possible edges. Characteristics of the considered graphs are presented in Table 1.

Graphs	Nodes	Edges	Hosting needed	Density
G1	15	14	573	13%
G2	14	18	581	20%
G3	11	17	433	30%
G4	17	54	791	40%
G5	18	77	880	50%
G6	13	47	589	60%
G7	15	74	652	70%
G8	17	109	737	80%
G9	12	59	428	90%
G10	16	120	687	100%

TABLE 1
Characteristics of generated graphs

Architecture-based compositions of services may be represented by sparse or dense graphs. So to experiment FB and FBR algorithms for architecture-based compositions of services, the density is an important criterion to consider. In Table 1, we have covered different densities of graphs to represent different types of compositions of services. Especially, we have considered 10 different densities from 10% to 100%. For each density level, we have randomly generated a graph.

Experiments we have already conducted in [10] have showed that the FB algorithm (see Section 4.2) gives good results for sparse graphs. We conducted more than 1000 experiments that showed that the FBR algorithm gives good results for not only sparse graphs but also dense graphs. All obtained results of the FBR algorithm were better than those obtained by the FB algorithm. For illustration, we present in the following some of these results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

5.1.1 Varying HQ

Among the above mentioned 10 graphs, we consider here three: one complete graph (Figure 5), one dense graph (Figure 6) and one sparse graph (Figure 7). In addition, we consider 18 different values of HQ (from 5% to 95 % of the hosting quantity of the considered graph).

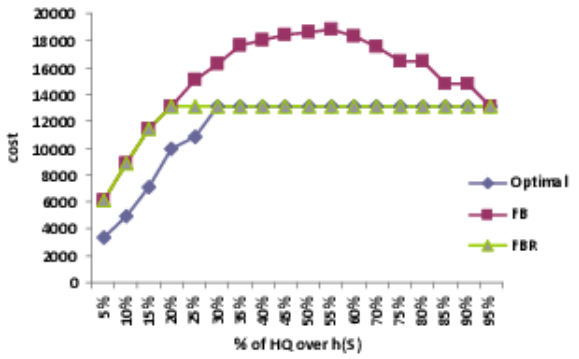


Fig. 5. Experiments on graph G10 ($\alpha = 15, \beta_1 = 10$ *and* $\beta_2 = 1$)

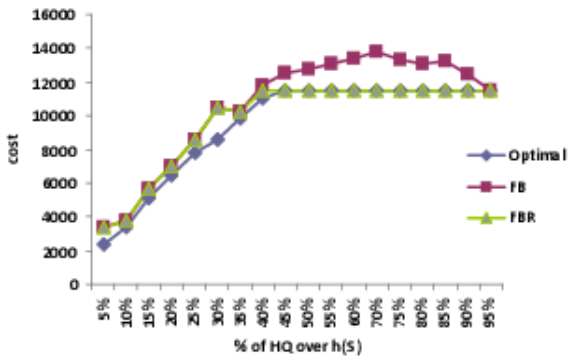


Fig. 6. Experiments on graph G7 ($\alpha = 15, \beta_1 = 10$ *and* $\beta_2 = 1$)

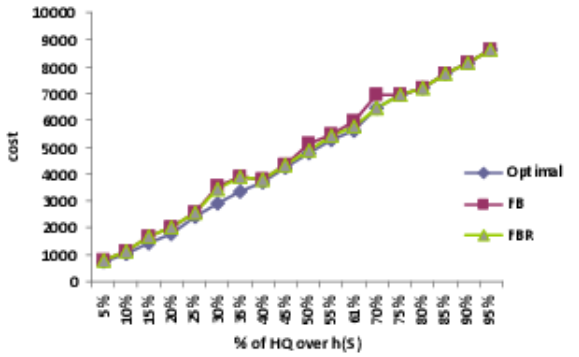


Fig. 7. Experiments on graph G1 ($\alpha = 15, \beta_1 = 10$ *and* $\beta_2 = 1$)

Figures 5, 6 and 7 show that our solutions given by the FBR algorithm is very close to optimal solutions. In addition, they show that while the FB algorithm behaves better for the sparse graph (Figure 7) than for the dense graphs (Figures 5 and 6), our new FBR algorithm has a good behaviour for all types of graphs.

For sparse graphs the FBR algorithm improves to some extent the FB algorithm performance thanks to the refinement procedure. This is valid whatever the HQ values (for example, see Figure 7).

For dense graphs, the FBR algorithm improves to some extent the FB algorithm performance when the HQ is small compared to the hosting value of the application (20% in Figure 5 and 45% in Figure 6) while the FBR algorithm behaves largely better than the FB algorithm when the HQ represents a big part of the hosting value of the application.

5.1.2 Varying types of graphs and hosting costs

Figure 8 presents the experiments on the 10 graphs (with different percentage of density) presented in Table 1. Over different values from 6 to 25 that α takes, the figure presents the average of the percentage of loss of FB and FBR algorithms compared to the optimal solution.

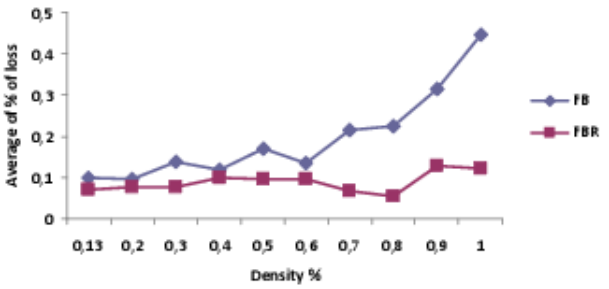


Fig. 8. Average over $\alpha \in \{6, 8, 12, 15, 20, 25\}$) of loss percentages ($\beta_1 = 10, \beta_2 = 1$)

For sparse graphs (density up to 60%, see Figure 8), the FBR algorithm performs better than the FB algorithm thanks to the refinement procedure that is used to tune the result of the Forward and Backward procedures that are used in the FB algorithm.

As for dense graphs (from 60% to 100%, see Figure 8), we found out that the combination backward-refinement is the best combination that is used in the FBR algorithm. This combination is behind the good results the FBR algorithm.

5.1.3 Varying graphs, hosting costs and HQ

Figure 9 (respectively Figure 10) presents experiments on the FB (respectively FBR) algorithm. We have considered the 10 graphs presented in Table 1 and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

varied HQ (from 5% to 95% of the hosting of each graph) and hosting costs from 6 to 25. Again, this implies that α varied according to β_1 and vice versa.

Figure 9 shows that for more that 17% of cases the FB reaches an optimal solution. For 57% it reaches an excellent result (the loss percentage does not exceed 15%).

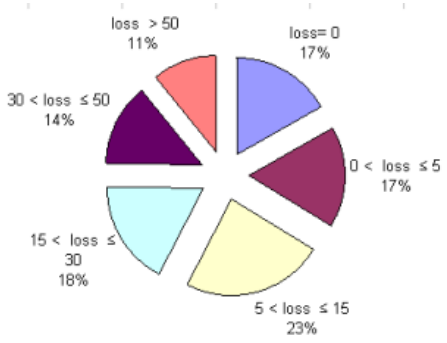


Fig. 9. FB algorithm: average of loss percentages (varying graphs, α and HQ)

Figure 10 shows that for more that 52% of cases the FBR reaches an optimal solution. For 82% it reaches an excellent result (the loss percentage does not exceed 15%).

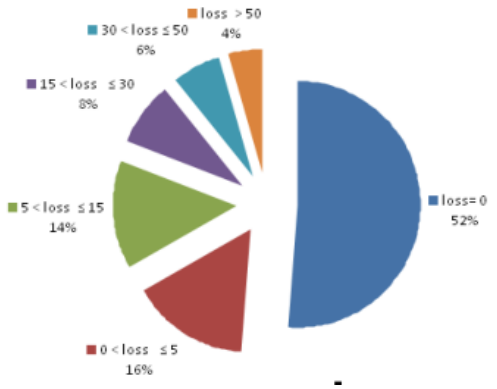


Fig. 10. FBR algorithm: average of loss percentages (varying graphs, α and HQ)

5.1.4 Conclusion

The results of these experiments are globally acceptable for FB algorithm. In fact, they are even excellent for sparse graphs (see the three first points in the FB curve of Figure 8). Whereas, the experiments show that the results of our new FBR algorithm are excellent for the different types of the considered graphs. It should be noted that compared to the FB algorithm, our new FBR algorithm reduces the average of the percentage of loss by a half for dense graphs.

These above experiment results show that the FB algorithm has a good performance for sparse graphs of services and has not reasonable performance for dense graphs of services while they show that the FBR algorithm has good performance for dense and sparse graphs. Since architecture-based compositions of services can be sparse or dense graphs, we can conclude that the FBR algorithm has a better performance than the FB algorithm for architecture-based compositions.

In the next sub-section, we present experiments of the FB and FBR algorithms on graphs that represent behavior-based composition described as structural processes.

5.2 Experiments on behavior-based compositions

Behavior-based compositions of services are generally sparse graphs where nodes represent services and operators and links represent dependencies between services and operators. It is obvious that the density here is not an important criterion to consider. But what matters is the structure that represents the Behavior-based compositions of services. Therefore, to evaluate the FBR Algorithm, we have developed a generator that guarantees that generated graphs are structured processes. Note that structured processes represent a class of behavior-based compositions largely used in industry and BPM community.

The generated graphs have a number of services between 5 and 11 and a number nodes between 9 and 19. We have considered 15 randomly generated SBA graphs which can reflect behavior-based compositions of services.

Characteristics of the considered graphs are presented in Table 2. Nodes include services and operators (And-Split, And-Join, Or-Split and Or-Join nodes). The generation of these graphs is based on the definition 1. The hosting needed is calculated as the sum of the hosting needed of all the nodes. The hosting needed for a node is randomly generated (less that 50).

Graphs	Nodes	Services	Hosting needed	Density
G5_2_9	9	5	135	25%
G5_1_11	11	5	141	21,82%
G6_1_14	14	6	119	16,48%
G6_2_14	14	6	145	16,48%
G7_1_11	11	7	166	21,82%
G7_2_13	13	7	140	20,51%
G7_3_15	15	7	220	17,14%
G8_1_12	12	8	227	19,70%
G8_2_16	16	8	123	14,17%
G9_2_12	12	9	332	22,73%
G9_1_17	17	9	264	14,71%
G10_1_16	16	10	225	16,67%
G10_2_16	16	10	194	16,67%
G10_3_18	18	10	167	15,03%
G11_3_19	19	11	275	14,04%

TABLE 2
Characteristics of generated graphs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

We conducted more than 1500 experiments that show that our FBR algorithm gives very good results. In the following, we present some of these results for illustration.

5.2.1 Varying HQ

Among the 15 graphs, we consider here three graphs with different number of nodes and services. In addition, we consider 18 different values of HQ (from 4% to 95% of hosting quantity of each considered graph).

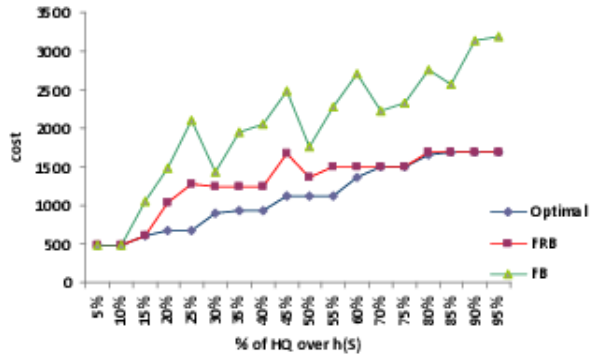


Fig. 11. Experiments on graph G8_1_12 ($\alpha = 6, \beta_1 = 10$ and $\beta_2 = 1$)

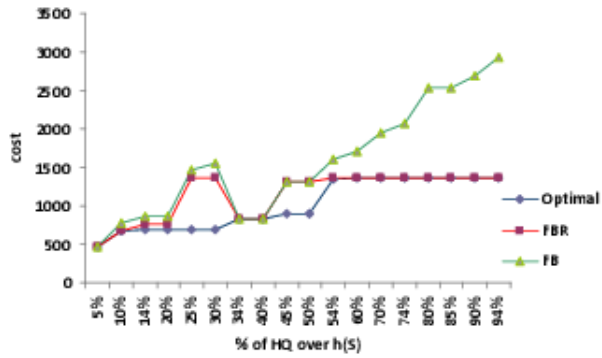
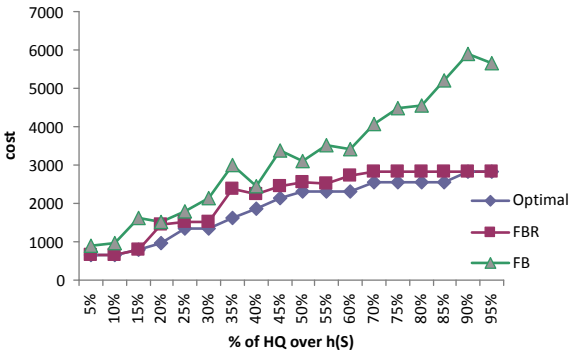


Fig. 12. Experiments on graph G6_2_14 ($\alpha = 6, \beta_1 = 10$ and $\beta_2 = 1$)

Figures 11, 12 and 13 show that our FBR algorithm gives solutions which are close to optimal solutions and better than those given by the FB algorithm. These results are similar with those relative to the other graphs of Table 2.

These experiments show that for behavior-based applications, when the percentage of the requested hosting quantity to be deployed in the public cloud is up to 60% (see Figures 11, 12 and 13), the FBR algorithm performs better to some extent than the FB algorithm thanks to the $Refinement$ procedure that is used to tune the result of the $Forward$ and $Backward$ procedures that are used in the FB algorithm. As for the cases where the requested hosting quantity



.eps

Fig. 13. Experiments on graph G10_2_16 ($\alpha = 12, \beta_1 = 10$ and $\beta_2 = 1$)

is from 60% to 100% of the hosting quantity of the application, we found out that the FBR algorithm is largely better that the FB algorithm thanks to the combination $backward-refinement$ that was the best combination among the three combinations that form the FBR algorithm.

5.2.2 Varying hosting costs

We consider in Figure 14 costs relative to graph $G8\_1\_12$ regarding the variation of α. Again, this shows that our FBR algorithm gives solutions which are close to optimal solutions and better than those given by the FB algorithm. These results are similar with those relative to the other graphs presented in Table 2.

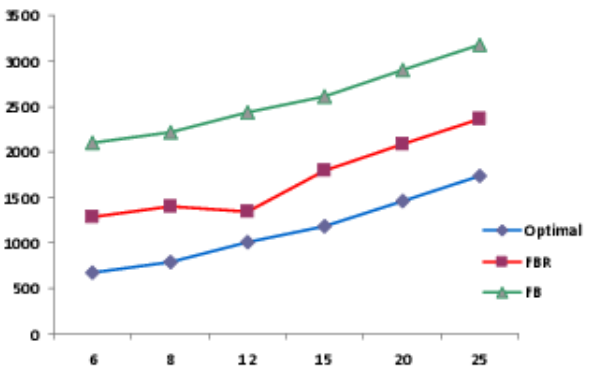


Fig. 14. Experiments on graph G8_1_12 ($HQ = 25\%, \beta_1 = 10$ and $\beta_2 = 1$)

5.2.3 Varying types of graphs and hosting costs

Figure 15 presents the experiments on the 15 graphs (with different percentage of density). It calculates the average of percentage of loss (over the different values from 6 to 25 that α takes). α is sometimes greater than β_1 and some times β_1 is greater than α. This is to ensure experiments that cover different situations of α compared to β_1 and vice versa.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2396076, IEEE Transactions on Services Computing

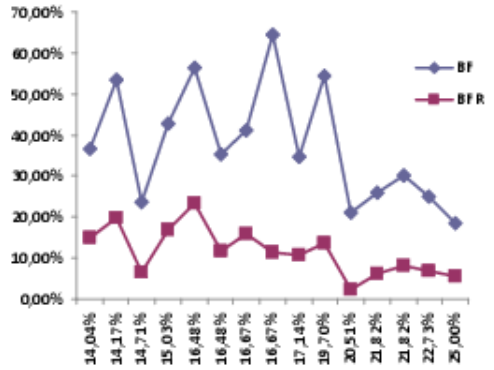


Fig. 15. Average over $\alpha \in \{6, 8, 12, 15, 20, 25\}$) of percentages of loss ($\beta_1 = 10, \beta_2 = 1$)

5.2.4 Conclusion

Our proposed FBR algorithm is a greedy algorithm and has a linear complexity (in time). Compared to the FB algorithm, our new FBR algorithm reduces the average of the percentage of loss by two-thirds for graphs that represent behavior-based compositions. The experiments show that the results of our new FBR algorithm are excellent for the different types of the considered graphs. The average of percentage of loss in Figure 8 does not exceed 9%. The percentage of loss of the worst case we met, among 1500 cases, is 12%.

6 Conclusion and Future Work

We tackled in this paper a NP-hard problem related to the bursting of a service-based applications to be deployed in hybrid clouds. This consists in determining a partition of the set of services composing the application while minimizing the hosting and communication costs. The partition is composed of two subsets: $Private$ and $Public$. Services in $Private$ are to be deployed locally in the private cloud and services in $Public$ are to be deployed in the public cloud.

In [10], we have presented the FB algorithm and we have proposed in this paper a new algorithm with acceptable complexity. To evaluate these two algorithms, we have conducted 2700 experiments. We have developed a parameterized graph generator that provides us with different realistic graphs that cover architecture-based compositions and behavior-based compositions described as structural processes.

Experiments results show that our FB algorithm has a good behavior, especially for sparse graphs which better represent service-based applications described as bihavior-based compositions. In addition, they show that our FRB algorithm has a good behavior not only for architecture-based compositions but also for those based on behavior. Compared to the FB algorithm, our new FBR algorithm reduces the average of the percentage of loss by a half for graphs that represent architecture-based compositions and by two-thirds for graphs that represent behavior-based compositions.

The work we achieved is very promising and several perspectives are under study. In this paper, we consider hosting and communications costs as criteria for service bursting of SBAs in hybrid clouds. In our future work, we will consider additional parameters such as security and privacy.

References

[1] B. Rimal, E.Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *International Joint Conference on INC, IMS and IDC*, 2009.

[2] I. Foster, Y. Z. ans I. Raicuand, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *The IEEE Grid Computing Environments*, ser. GCE'08, Austin, USA, 2008.

[3] X. Zhang, X. Chen, Y. Zhang, Y. Wu, W. Yao, G. Huang, and Q. Lin, "Runtime model based management of diverse cloud resources," in *MoDELS'13*, 2013, pp. 572–588.

[4] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen, "Distributed automatic service composition in large-scale systems," in *Proceedings of the Second International Conference on Distributed Event-based Systems*, ser. DEBS '08. New York, NY, USA: ACM, 2008, pp. 233–244. [Online]. Available: http://doi.acm.org/10.1145/1385989.1386019

[5] J. Marino and M. Rowley, *Understanding SCA (Service Component Architecture)*, 1st ed. Addison-Wesley Professional, 2009.

[6] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[7] M. B. Juric, *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition.*, P. Publishing, Ed. ISBN:1904811817, 2006.

[8] O. M. G. (OMG), "Business process model and notation (bpmn) version 2.0," Object Management Group (OMG), Tech. Rep., jan 2011. [Online]. Available: http://taval.de/publications/BPMN20

[9] A. S. Fadel and A. G. Fayoumi, "Cloud resource provisioning and bursting approaches," in *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2013, Honolulu, Hawaii, USA, 1-3 July, 2013*. IEEE, 2013, pp. 59–64. [Online]. Available: http://dx.doi.org/10.1109/SNPD.2013.2

[10] F. B. Charrada, N. Tebourski, S. Tata, and S. Moalla, "Approximate placement of service-based applications in hybrid clouds," in *WETICE*, S. Reddy and K. Drira, Eds. IEEE Computer Society, 2012, pp. 161–166.

[11] J. Famaey, T. Wauters, F. D. Turck, B. Dhoedt, and P. Demeester, "Towards efficient service placement and server selection for large-scale deployments," in *Proceedings of the 2008 Fourth Advanced International Conference on Telecommunications*, ser. AICT '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 13–18.

[12] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Comp. Syst.*, vol. 28, no. 2, pp. 358–367, 2012.

[13] A. Sailer, M. R. Head, A. Kochut, and H. Shaikh, "Graph-based cloud service placement," in *IEEE SCC*. IEEE Computer Society, 2010, pp. 89–96.

[14] T. Kwok and A. Mohindra, "Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications," in *Proceedings of the 6th International Conference on Service-Oriented Computing*, ser. ICSOC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 633–648.

[15] J. Lucas-Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments," in *Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011)*, July 2011, pp. 1–7.

[16] ——, "Scheduling strategies for optimal service deployment across multiple clouds," *Future Generation Computer Systems*, vol. in press, 2012.

[17] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 228–235. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2010.58

[18] S. Chaisiri, B. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *4th IEEE Asia-Pacific Services Computing Conference, IEEE APSCC 2009, Singapore, December 7-11 2009, Proceedings*, M. Kirchberg, P. C. K. Hung, B. Carminati, C. Chi, R. Kanagasabai, E. D. Valle, K. Lan, and L. Chen, Eds. IEEE, 2009, pp. 103–110. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5394134&isnumber=5394078

[19] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Gener. Comput. Syst.*, vol. 28, no. 2, pp. 358–367, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.07.003

[20] S. Phani Praveen, U. Tulasi, and K. Ajay Krishna Teja, "A cost efficient resource provisioning approach using virtual machine placement," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 2, pp. 2365–2368, Mar. 2014.

[21] E. Wohlstadter, N. Kaviani, and R. Lea, "Manticore: A framework for partitioning software services for hybrid cloud," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, ser. CLOUDCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 333–340. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2012.6427541

[22] C. H. Papadimitriou, "On the complexity of integer programming," *J. ACM*, vol. 28, no. 4, pp. 765–768, Oct. 1981. [Online]. Available: http://doi.acm.org/10.1145/322276.322287

[23] B. Kiepuszewski, A. H. M. t. Hofstede, and C. Bussler, "On structured workflow modelling," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, ser. CAiSE '00. London, UK, UK: Springer-Verlag, 2000, pp. 431–445. [Online]. Available: http://dl.acm.org/citation.cfm?id=646088.679917

[24] The Apache Software Foundation, *Getting started with Tuscany*, http://tuscany.apache.org/getting-started-with-tuscany.html.

[25] S. Sahni, "Computationally related problems," *SIAM Journal on Computing*, vol. 3, pp. 262–279, 1974.