The Journal of Systems and Software 134 (2017) 138–152

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Multi-cloud service composition using Formal Concept Analysis

Haithem Mezni [a,*], Mokhtar Sellami [b]

[a] SMART Lab/University of Jendouba, Jendouba, Tunisia
[b] Higher Institute of Technological Studies of Jendouba/RIADI Lab, Tunisia

ARTICLE INFO

Article history:
Received 17 February 2017
Revised 4 July 2017
Accepted 8 August 2017
Available online 9 August 2017

Keywords:
Cloud computing
Multi-cloud
Service composition
Lattice theory
Formal concept analysis

ABSTRACT

Recent years have witnessed a rapid growth in exploiting Cloud environments to deliver various types of resources as services. To improve the efficiency of software development, service reuse and composition is viewed as a powerful means. However, effectively composing services from multiple clouds has not been solved yet. Indeed, existing solutions assume that the services participating to a composition come from a single cloud. This approach is unrealistic since the other existing clouds may host more suitable services. In order to deliver high quality service compositions, the user request must be checked against the services in the multi-cloud environment (MCE) or at least clouds in the availability zone of the user. In this paper, we propose a multi-cloud service composition (MCSC) approach based on Formal Concept Analysis (FCA). We use FCA to represent and combine information of multiple clouds. FCA is based on the concept lattice which is a powerful mean to classify clouds and services information. We first model the multi-cloud environment as a set of formal contexts. Then, we extract and combine candidate clouds from formal concepts. Finally, the optimal cloud combination is selected and the MCSC is transformed into a classical service composition problem. Conducted experiments proved the effectiveness and the ability of FCA based method to regroup and find cloud combinations with a minimal number of clouds and a low communication cost. Also, the comparison with two well-known combinatorial optimization approaches showed that the adopted top-down strategy allowed to rapidly select services hosted on the same and closest clouds, which directly reduced the inter-cloud communication cost, compared to existing approaches.

1. Introduction

Service-oriented computing and cloud computing have a reciprocal relationship that allows to provide consumers with the "computing of services" and the "services of computing" (Wei and Blake, 2010). Cloud environment becomes a natural choice to deliver various types of resources as services.

To satisfy user needs, Cloud service-based systems are often designed by invoking several providers. Cloud service composition allows integrating various existing cloud resources into a set of interacting services to deliver Cloud-based solutions that meet specific quality criteria (Jula et al., 2014).

Since 2009, several service composition methods have been proposed in the context of Cloud computing. However, most of them assume that all services involved in a composition come from a single cloud, instead of finding services from multiple availability zones. This assumption is unrealistic and limits the benefits received from other clouds that might deploy better services (Microsoft Communications & Media Industries, 2013). For example, some clouds are well suited for services that are frequently accessed by cloud consumers who require a certain data transfer rate. Also, the services delivered to a consumer may belong to a Cloud infrastructure that does not satisfy certain requirements, such as security and compliance requirements or business and technical needs.

Multi-cloud approach refers to cloud services consumed through several cloud providers (Ardagna et al., 2012; Petcu, 2011). To guarantee high availability and quality levels for their offered services and to mitigate risk from data loss or downtime, organizations often deploy their services using several cloud providers in their availability zones (Venkat, 2016). This multi-cloud strategy brings many benefits such as improved enterprise performance and lower costs. This also permits to avoid "vendor lock-in" and makes use of different infrastructures to offer the same services for different users' profiles. That is why leading cloud providers, such as IBM and Microsoft, decided to develop platforms and address the challenges of multi-cloud service provisioning (Microsoft Communications & Media Industries, 2013).

* Corresponding author.
E-mail addresses: haithem.mezni@gmail.com, haithem.mezni@fsjegj.rnu.tn (H. Mezni), sellamimokhtar@yahoo.com (M. Sellami).

http://dx.doi.org/10.1016/j.jss.2017.08.016
0164-1212/

However, combining the distributed services across multiple clouds raises several issues, such as the inter-cloud communication cost, the increase of financial charges, the security and privacy issues, etc. Hence, a challenging task is to minimize the number of participating clouds while considering services' constraints.

The goal of this work is to offer optimal service compositions by considering multiple cloud providers. The major difference between our approach and existing solutions is that we, not only aim to reduce the number of clouds that participate to a composition, but also the number of service providers within the minimal set of selected clouds. Another difference is to consider the inter-clouds communication time so that to reduce the total execution time of a composed service.

In this paper, a solution based on Formal Concept Analysis (FCA) method is proposed to find a minimal set of clouds that host the optimal service composition, while reducing the inter-cloud communication cost. We use lattice theory to regroup multi-cloud objects (i.e. clouds, providers, services) into small clusters based on their common properties. Then, we exploit the hidden associations between these clusters to extract valuable information (e.g. common hosting clouds, communication costs) for effectively and rapidly finding the best combination in the multi-cloud space. The regrouping capabilities of FCA will permit to group services according to their providers and also to group those providers that belong to the same clouds.

FCA has been adopted to solve several research problems including Web service selection (Fenza and Senatore, 2010), information retrieval (De Maio et al., 2012), ontology construction (Weng et al., 2006), knowledge extraction (Poelmans et al., 2013a), machine learning (Kuznetsov, 2004), etc. In the context of Cloud and Big data environments, FCA has been successfully applied to deal with large-scale problems, such as stream processing in smart cities (De Maio et al., 2017), modelling of big medical data (Hao et al., 2016) and continuous k-nearest neighbor search in complex road networks (Ferchichi and Akaichi, 2016). This successful adoption suggests that FCA may be a good candidate to the multi-cloud service composition problem.

The main contributions of this paper are as follows:

- *Modelling the multi-cloud environment using concept lattice theory:* we describe the MCE using two types of formal contexts, from which a set of complete concept lattices is generated to group the providers according to their offered services, and to express the relations between providers and their hosting clouds.
- *Defining algorithms for the extraction of candidate providers and clouds.* An algorithm is also defined to determine the optimal cloud combination.
- *Designing a system model* that allows exploring the multi-cloud base and delivering the minimal set of providers and clouds that satisfy the user request.
- *Implementing the FCA-based service composition approach* and experimentally evaluating the quality of produced solution.

The rest of this paper is organized as follows. Section 2 details the related works to the multi-cloud service composition (MCSC). Section 3 formulates the MCSC problem. Section 4 provides the basic notions of Formal Concept Analysis. Section 5 shows how the MCE is modelled using FCA. Section 6 presents our approach and the way FCA is applied to the multi-cloud service composition problem. Section 7 provides a theoretical complexity study of the proposed MCSC method. Section 8 presents the implementation details, the conducted experiments and a discussion of the results. Section 9 is devoted to the conclusion and the future work.

2. Related work

A common assumption among existing approaches was that all the services that participate to a composition belong to the same cloud. Extensive surveys have been presented for Web service composition (Sheng et al., 2014) and service composition in single clouds (Jula et al., 2014). However, very few approaches have been proposed to deal with service composition across multiple clouds. The collected works from the literature do not exceed eight approaches, which will be discussed in this section.

The first work was proposed by Zou et al. (2010). The authors use the tree structure to model the multi-cloud base (MCB). Then, a minimum request set is returned by searching the MCB tree. To select the optimal cloud set with a feasible composition sequence, the authors proposed three cloud combination algorithms. The "All Clouds Combination" algorithm considers all clouds as inputs for the composition and enumerates all possible solutions. This method locates a service composition sequence in a short execution time, but using a high number of clouds. The second algorithm "Base Cloud Combination" recursively finds a service composition in the whole cloud combinations set. However, this algorithm requires a high execution time to find the optimal cloud combination. The last algorithm aims to determine a near-optimal cloud combination based on an approximation algorithm. However, it remains time consuming and it may fail to find the optimal cloud combination because it obtains the combination of clouds using the services files, which could constrain the combinations.

In Gutierrez-Garcia and Sim (2013), an agent-based approach to composing services in multi-cloud environments has been proposed. Gutierrez-Garcia and Sim considered various types of services (e.g., one-time virtualized service, persistent virtualized services, vertical and horizontal services). They used a semi-recursive contract net protocol and endowed the composition agents with catalogs containing information about cloud participants. Despite the promising empirical results, this solution inherits the limitations of the agent-based distributed approach (i.e. processing and communication costs).

Collective intelligence was adopted by Yu et al. (2015). Based on Ant Colony Optimization (ACO), the authors used two cloud combinations algorithms: the Greedy-WSC and the ACO-WSC (ACO for web service composition) to obtain a valid services composition that uses the minimum number of clouds. In the first one, they aim to select the cloud that contains the high number of required services. Whereas in the second, artificial ants try to find the optimal graph's path according to the pheromone amount and the heuristic information on the edges, knowing that each graph's node represents one cloud and each edge is a connection between two different clouds.

Kurdi et al. (2015) propose a combinatorial optimization algorithm for cloud service composition (COM2) that aims to efficiently compose services with small numbers of examined services and combined clouds. The proposed algorithm selects the cloud with the maximum number of services to increase the possibility of fulfilling service requests with minimal overhead costs.

To deal with various versions of quality of service (QoS) information revealed in different mobile CPS applications and to reduce cost and time of consumption, the authors in Wu et al. (2014) propose a cloud service selection method named CSSM. It aims at mining qualified versions of cloud services for cross-cloud service composition. The proposed approach takes the utility value as the evaluation index and aims at finding an optimal trusted composition from a Cloud services set using an extended top-k iteration composition process.

To support cross-platform service invocation in cloud environment, Qi et al. (2012) presented a QoS-aware composition method named LOEM. This method aims at improving the quality of pro-

vided compositions when a large number of composite solutions is available in cloud environment. Also, a decision-making method is proposed to determine whether a QoS-aware WSC problem has a QoS qualified composite solution. Although the proposed approach provided near-optimal compositions, the time complexity of LOEM is still exponential, which cannot deliver satisfactory results in case of real-time user requests.

To deal with privacy issues in cross-cloud service composition, an enhanced history record-based service optimization method, named HireSome-II, has been developed by Dou et al. (2015) for processing big data applications. To enhance the credibility of a composition plan, the authors propose to evaluate a service by some of its QoS history records, rather than its advertised QoS values. Also, the k-means algorithm is introduced into HireSome-II as a data filtering tool to select representative history records.

Parallel programming models were also adopted to optimize service composition in large-scale cloud mashup applications. Zhang et al. (2016) propose a skyline query processing method that aims to construct multi-cloud mashup applications based on the quality of mashup services and quality of user experience. To achieve the skyline selection goals, the authors use the MapReduce parallel programming model in order to process the different mashup cloud platforms. A partitioning method based on block elimination is also proposed to reduce the search space and optimize the mashup composition scheme.

Other related issues have been addressed in the context of multi-cloud environments. Jrad et al. (2015) studied SLA matching of scientific workflows properties in multiple Clouds. They proposed an ontological model for the semantic description of composite multi-cloud services. Also, a utility-based genetic algorithm was proposed to select optimal Cloud resources based on the matching between user requirements and Clouds properties.

In Kritikos and Plexousakis (2015), Kritikos and Plexousakis aim to design multi-cloud applications through the composition of Cloud services. The authors propose an abstract deployment model to specify a set of requirements, including resource, QoS, location, security and cost. The model is then concretized by considering the set of cloud provider offerings and internal SaaS realizations of the requester organization. To optimize the design of multi-cloud applications, the authors also combine a constraint satisfaction optimization problem (CSOP) technique and Constraint Programming with interval arithmetics. Choco and Ibex are exploited as constraint solving engines.

From the above discussed works, we came to the following conclusions:

- None of the existing approaches have considered the communication cost between clouds and only focus on reducing the number of clouds. Our approach solves this problem by modelling the relations between clouds and by considering the inter-clouds communication in the selection of the optimal clouds.
- Existing approaches only examine the clouds with high number of services to reduce the composition time, while it is important to consider the services in all the candidate clouds during the composition process. Our approach exploits the powerful grouping capabilities of Formal Concept Analysis to extract the relevant providers, which will be used after to determine the minimal cloud set.

Based on the above discussions, our work aims to resolve the major issues related to exploring the multi-cloud environment as well as reducing, not only the number of clouds, but also the inter-cloud communication cost. This goal is guarantee by exploiting the strengths of Formal Concept Analysis as a powerful mean of clustering the clouds and services related information.

3. Problem formulation

In this work, the problem of multi-cloud service composition is defined as follows:

"Given a set of clouds hosting the services offered by a number of providers, the goal is to determine the minimal and cost effective sub-set of clouds, from which an optimal service composition is selected".

The set of clouds, also called multi-cloud environment (MCE), forms a big distributed container of services and their related data. Each cloud is an independent service repository.

Definition 1 (Multi-cloud environment). a MCE is a set $C=\{C_1, C_2, \ldots, C_N\}$ of Clouds, where C_i $(1 \leq i \leq N)$ is a cloud that hosts a set P of providers, $P=\{P_{i1}, P_{i2}, \ldots, P_{iM}\}$, where P_{ij} $(1 \leq j \leq M)$ is the jth provider in cloud C_i. A provider may belong to more than one cloud. A provider offers a set S of services, $S=\{S_{j1}, S_{j2}, \ldots, S_{jL}\}$, where S_{jk} $(1 \leq k \leq L)$ is the kth service offered by provider P_j.

To model the communication network between clouds, we consider the MCE as a directed acyclic graph $G=(V, E)$, where V is the set of vertices (clouds in our case) and $E \in V \times V$ is the set of edges. E denotes the set of communication paths transmitting information between two Clouds C_i and C_j. The cost of each communication path is represented by $E_{ij}: E \rightarrow \mathbb{R}^+$.

In this work, a composition of selected services is a sequence of tuples, $W=\{\langle S_1, C_1\rangle, \langle S_2, C_2\rangle, \ldots, \langle S_k, C_k\rangle\}$, where $S_1, S_2, \ldots, S_k$ are the services involved in the composition, and $C_1, C_2, \ldots, C_k$ denote the minimum Cloud set. Based on this statement, we may regard the selection of optimal cloud combination as the traditional sub-graph matching problem (Cheng et al., 2008), where the whole graph is the MCE and the sub-graph is the minimum set of Clouds. As a result, a service query is used to determine the candidate providers which will serve to extract the sub-graphs of cloud combinations.

Definition 2 (Service-query). A user request S_r is denoted by a graph $G_r=(S', E')$, with $S'=\{S_1', S_2', \ldots, S_n'\}$ is a set of requested services, and $E'=\{E_1', E_2', \ldots, E_m'\}$ is the set of possible data flows between requested services. There exists a service composition $W=(S, CP)$ in the multi-cloud repository, which satisfies S_r if and only if $S' \in S$ and $E' \in CP$, where CP denotes a communication paths between two clouds or within a same cloud.

The next section presents the basic notions of FCA and their intended used in the multi-cloud service composition approach.

4. Applying Formal Concept Analysis

As a branch of lattice theory, Formal Concept Analysis (FCA) is a clustering technique for knowledge representation, data analysis, and information management (Poelmans et al., 2013b). FCA is a lattice-theory paradigm for discovering conceptual structures in data. Since these structures allow the extraction of dependencies within the data, we exploit FCA to discover the hidden relations between clouds and services related information.

The basic structures of FCA are presented by the following definitions:

Definition 3 (Formal context). A formal context is a triple $K=(O, A, R)$ denoting respectively the set of objects, the set of attributes, and the set of binary relations between the objects and the attributes $(R=O \times A)$. An object "o" has the attribute "a" if o and a are in relation R (denoted by oRa).

Definition 4 (Formal concept). A formal concept of the context $K=(O, A, R)$ is a pair of sets (E, I) such that $E \subseteq O$ and $I \subseteq A$. A formal concept is a cluster with two parts: the *Extent* part E groups

Table 1
Example of multi-cloud environment and its providers/services distribution.

Clouds	C1					C2			C3			C4			C5		
Providers	P1	P2	P3	P4	P5	P5	P8	P9	P2	P3	P7	P6	P8	P10	P3	P4	P6
Services	5	6	6	4	4	4	11	3	6	4	5	2	11	4	6	4	2

the objects that share a sub-set of common attributes. These latter are stored in the *Intent* part I of the formal concept. Elements in I denote the common attributes shared by all the objects in E.

Given a formal context K, we derive a set of "formal concepts" using derivation operators. The set of all concepts is ordered by the sub-concept/super-concept relation to make a complete hierarchical structure called "concept lattice".

Definition 5 (Derivation operators). Let $K=(O, A, R)$ be a formal context and $O' \subseteq O$ be a set of objects. We define $A' = \{a \in A \mid \forall o \in O'$: oRa$\}$, i.e. A' is the set of all attributes that all objects in O' share. Analogously, let $A'' \subseteq A$ be a set of attributes. We define $O'' = \{o \in O \mid \forall a \in A''$: oRa$\}$; i.e. O'' is the set of those objects that have all attributes from A''.

The major motivation behind using FCA in the multi-cloud composition problem is its mathematical foundations and soundness. Moreover, FCA is based on rich stack of incremental algorithms to construct and modify concept hierarchies.

The "lattice" structure in FCA depicts the notion of concepts that regroup the objects sharing the same attributes. In the same fashion, we use FCA to regroup clouds that host the same providers and providers that offer similar services. Also, using the Galois sub-hierarchy (GSH) allows us to reduce the time devoted to compute the minimal set of Clouds that covers the user query. One of the strengths of GSH is its ability to eliminate redundant concepts. This allows having optimal clusters of cloud objects (e.g., providers within same cloud(s), services offered by the same provider) and reducing the search space of minimal cloud set.

5. Characterizing multi-cloud environment using Formal Concept Analysis

The correct characterization of multi-cloud environment objects (clouds, providers, services) is the first step towards effective service composition. For this purpose, FCA is adopted to represent and analyse the services data in each Cloud. Such data are represented in the form of formal contexts.

We modelled the MCE as a set of formal contexts. Each cloud is described as a formal context (FC), from which a complete concept lattice is generated to group the providers according to their offered services. An additional formal context is also created to express the relations between providers and their hosting clouds.

The first FC called "Multi-Cloud Formal Context" (denoted K^{MC}) represents a complete multi-cloud environment and consists of a set of providers with their hosting clouds. This formal context indicates whether a provider P deploys its services S on the cloud C or not. The second formal context K^C called "Cloud Formal Context" describes the relationships between providers and their offered services.

In many cases, service providers may publish their services in different clouds. Hence, a provider may belong to more than one formal context. Given N clouds, the information regarding services and their corresponding providers are modelled in N formal contexts K^C, where each one represents a complete cloud. Candidate providers extracted from these formal contexts are, then, used to extract candidate clouds from the multi-cloud formal context K^{MC}.

Table 2
Formal context K^{MC} of providers in all the clouds.

MCE	C1	C2	C3	C4	C5
P1	1	0	0	0	0
P2	1	0	1	0	0
P3	0	0	1	0	1
P4	0	0	0	0	1
P5	0	1	0	0	0
P6	0	0	0	1	1
P7	0	0	1	0	0
P8	0	1	0	1	1
P9	0	1	0	0	0
P10	1	0	0	1	0

Definition 6 (Multi-cloud formal context). A multi-cloud formal context is a triple $K^{MC}=(P, C, R)$ where P is a set of objects representing service providers, and C is the set of attributes representing the hosting clouds. The binary relation $R = P \times C$ indicates whether a provider belongs to a cloud or not. An object 'p' has an attribute 'c' if p and c are in relation R (denoted by pRc). This means that provider 'p' deploys its services in the cloud 'c'.

Definition 7 (Cloud formal context). A cloud formal context is a triple $K^C=(S,P,R)$ where S is a set of objects representing available services, and P is the set of attributes representing the providers. The binary relation $R = S \times P$ indicates whether a service is offered by a provider or not. An object 's' has an attribute 'p' if s and p are in relation R (denoted by sRp). This means that service 's' is offered by the provider 'p'.

Take the example of multi-cloud environment in Table 1. 50 services with different functional and QoS capabilities are offered by 10 providers, which are hosted in 5 clouds. For example, the Cloud C1 hosts 5 providers that offer together 25 services. Note that some providers may deploy their offered services on several clouds (e.g., providers P2, P3, P4, P5, P6, and P8).

Based on the above example, the multi-cloud environment is modelled using five *Cloud formal contexts* K^C and a *Multi-cloud formal context* K^{MC}. From these formal contexts, we use FCA to produce hierarchically ordered clusters of providers offering the same services (see cloud lattice L^C in Fig. 2) and clusters of providers hosted in the same cloud (see multi-clouds' lattice L^{MC} in Fig. 1). In the literature, there are several algorithms that can be used to build the lattices L^{MC} and L^C. Examples include Bordat, Godin, Dowling, Next closure, Faster, etc. (Kumar and Singh, 2014). In our work, we use Bordat algorithm offered with Galicia[1] tool. This algorithm is suitable for large formal contexts and performs well on contexts of low and average density (Kuznetsov and Obiedkov, 2002). In a multi-cloud setting, providers cannot deploy their offered services in all (or a large number of) clouds due to the financial charges and the complexity of managing the huge number of services. Hence, the multi-cloud formal context is characterized by an average density.

Based on the MCE example in Table 1, the following formal context describes the relations between providers and their hosting clouds (Table 2).

[1] http://www.iro.umontreal.ca/~galicia/.

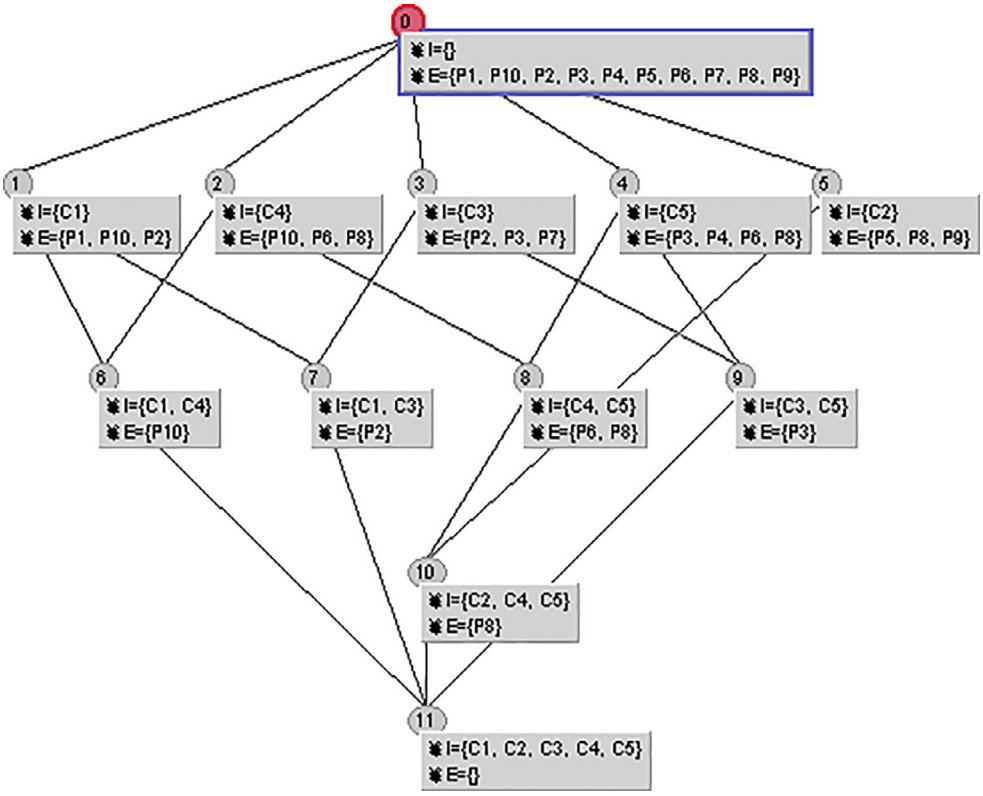

Fig. 1. Concept lattice of providers in all the clouds.

Table 3
Formal context K^C of providers and their offered services in the first cloud.

Cloud 1	P1	P2	P3	P4	P5
S1	1	0	1	0	0
S2	0	1	1	1	0
S3	0	0	1	0	1
S4	1	0	1	0	1
S5	0	1	0	0	0
S6	1	1	0	1	0
S7	0	0	1	0	1
S8	1	1	1	1	0
S9	0	1	0	0	1
S10	1	1	0	1	0

The set of formal concepts in Fig. 1 is derived from the above formal context to form a complete lattice L^{MC} of the multi-cloud environment. For example, the second formal concept {P1, P10, P2, C1} in L^{MC} shows that *Cloud 1* hosts the services of three providers. The extent part in the concept {P3, C3, C5} consists of two clouds hosting the same provider. This means that two candidate cloud combinations are available if the provider P3 is involved in a service composition request. Regarding the formal concept {P6, P8, C4, C5}, the extent part shows that both providers P6 and P8 deploy their services in the same clouds {C4, C5} specified in the Intent part of the concept.

One of the strength of FCA is its ability to group information in a way that allows deriving new knowledge. Indeed, the Intent parts in the lattice of services allow us to choose between several providers that offer the same service. In the same way, the Intent parts in the lattice of clouds allow to select one of the clouds that host the same provider, by taking into account the user's constraints, such as the availability zone and the required service execution time.

Take the first cloud in the MCE example. Table 3 shows a formal context describing the relations between services and their providers within this cloud. The entries in this formal context (i.e. binary relations) indicate whether a service is offered by a provider or not.

From the above formal context, the derived lattice L_1^C in Fig. 2 represents the hierarchy of concepts regrouping services according to their providers. For example, in the eleventh formal concept the Extent part consists of 3 services {S6, S8, S10}, while the Intent part comprises 3 providers {P1, P2, P4} offering those services. The 10th concept {S9, P2, P5} shows that two providers offer the same service S9.

Next, we describe the process of extracting candidate service providers and candidate clouds in order to find the optimal service composition.

6. FCA-based multi-cloud service composition

To reduce the communication costs between web services that come from different clouds, we aim in this paper to find the minimal combination of clouds that host the best service composition, while satisfying the constraints specified in the user request. We present three algorithms for selecting a feasible cloud combination that uses a minimum number of clouds, and for evaluating the candidate service compositions in the optimal cloud combination.

An overview of the proposed multi-cloud service composition model is illustrated in Fig. 1. This figure shows that the input of a composition problem is a service request and a set of formal contexts representing the multi-cloud repository. The output is an optimal service composition.

The steps of our approach are briefly described as follows:

- *Step 1 (Extraction of candidate providers):* this step extracts, from each cloud lattice, the candidate providers that offer the requested services. Each set of providers that satisfy the user request is used in the next step to determine the possible cloud combinations.

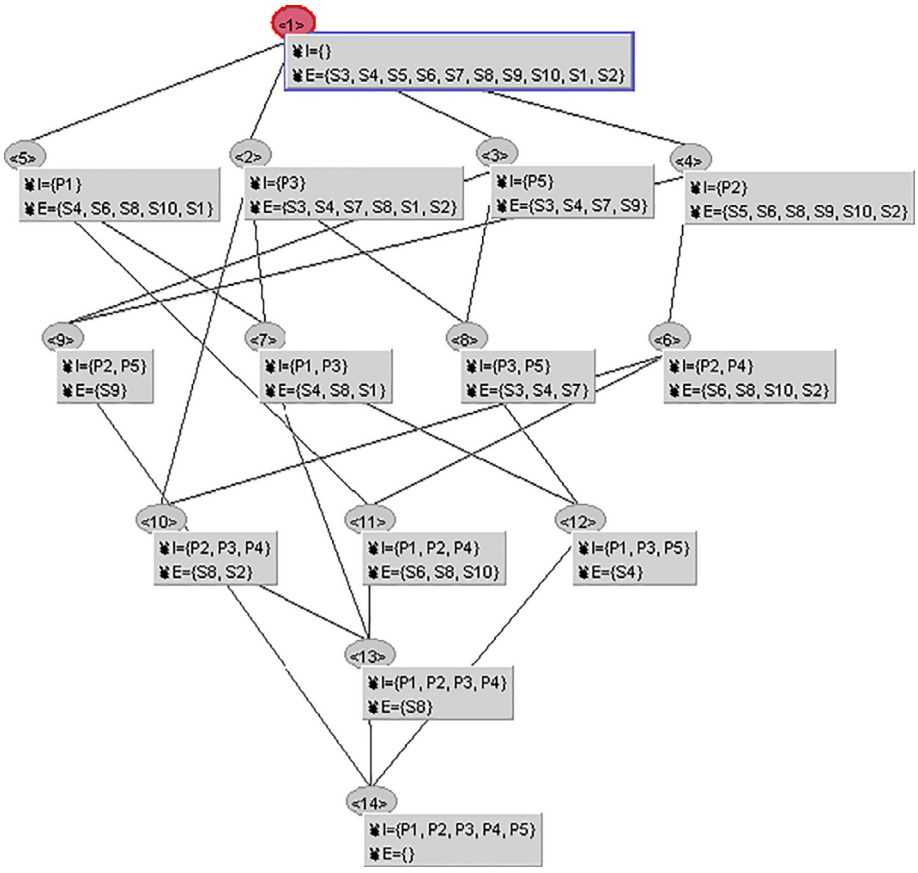

Fig. 2. Concept lattice of services available in Cloud 1.

- *Step 2 (Construction of candidate cloud combinations):* in this step, the lattice that represents the relations between service providers and their hosting clouds is used to filter the candidate clouds with respect to the selected providers in step 2. Since a provider may belong to one or more clouds, several combinations of clouds may contain the valid service composition. Those combinations are constructed according the communication links in the MCE.
- *Step 3 (Selection of optimal clouds):* this step consists of selecting the appropriate and minimal cloud set. This is achieved by considering the cloud combination's size and the inter-clouds communication cost.
- *Step 4 (Service composition):* in this final step, the multi-cloud service composition problem is transformed into a classical service composition. An algorithm is used to select the suitable services from the optimal cloud set.

The following sub-sections give more details about each of the above steps.

6.1. Extraction of candidate providers

This step exploits the regrouping capabilities of FCA to run through the formal concepts in each Cloud lattice L^C and to extract the useful concepts. To do that, the algorithm "Candidate providers' extraction" takes as input a user request S_r containing the required services, and N lattices L_i^C of services ($1 \leq i \leq N$), where each lattice L_i^C represents the set of services and their providers hosted in the ith cloud. The output of the algorithm is M sets of service providers SP_{ij} ($1 \leq j \leq M$), where each set represents the jth possible combination of providers in the ith cloud, that may together meet the user request.

We adopt a top-down approach to ensure that the formal concepts with the highest Extent sizes in each lattice L^C are processed firstly. This decreasing sorting helps to rapidly identify the providers with the greater number of services. The "Candidate providers' extraction" algorithm is described as follows.

The algorithm starts the process of determining the candidate providers in each cloud by running through the corresponding lattice L_i^C (line 3). For each concept C_j in the lattice, the algorithm checks if the *Extent* part contains some or all of the required services (lines 6 and 7). Then, the services found in the jth concept's Extent are added to the set S_a of found services (line 8), and the providers (Intent part of C_j) that offer these services are stored in the vector V_{prov} (line 9).

The search process is repeated until the selected providers cover all the required services, that is, until meeting the condition $S_a = S_r$ (line 11). At this stage of the algorithm, the candidate providers in V_{prov} are combined together (line 12), in order to construct the possible solutions that will be stored in the output providers' sets PS (line 13).

Since the remaining concepts in the lattice L_i^C may contain other candidate service providers, the whole process is continued (line 6) after initializing the used sets (S_a and V_{prov}). The whole routine is performed for each cloud Cl_i (line 17).

In what follows, we show how the possible providers' sets are exploited to extract the candidate cloud combinations and to select the optimal clouds.

6.2. Construction of candidate cloud combinations

This step of the multi-cloud service composition process aims to regroup the clouds that may together offer the required services. By evaluating each cloud combination, we may determine the op-

timal set of clouds from which the suitable services are delivered to the user.

The algorithm below takes as input a multi-cloud lattice L^{MC} and M sets of service providers PS_i $(1 \leq i \leq M)$, each one offers a possible composition of services. The output is a set *CCS* of cloud combinations; each one represents the group of clouds that may meet the user request.

We adopt a top-down approach to ensure that the formal concepts in the multi-cloud lattice are sorted in decreasing order of the Extent size. This decreasing sorting helps to rapidly identify the clouds with the greater number of providers, which allows reducing the number of participating clouds. The main steps of the proposed algorithm are summarized as follows.

For each group of candidate providers PS_i $(1 \leq i \leq M)$, the goal of the algorithm is to determine the combinations of clouds that host these providers. To do that, the algorithm runs through the multi-cloud lattice L^{MC} (line 5) and checks if the *Extent* part of the current concept C_j in the lattice contains some or all of the candidate providers (line 6). In this case, the corresponding hosting clouds, which belong to the *Intent* part of C_j are stored in the vector V_{comb} of candidate clouds (line 7). If no cloud is identified in the Intent part, the next formal concept is checked until appropriate clouds are obtained. Once located, they are added to the vector V_{comb}.

The search iteration of candidate clouds is broken when all the candidate providers are found (line 10). At this stage of the algorithm, the candidate clouds in V_{comb} are combined together (line 12), in order to construct the possible cloud combinations, that will finally be stored in the output set *CCS* (line 13).

The last step of our approach is devoted to the selection of the optimal cloud combination. This is achieved by considering several criteria, such as the number of clouds, the number of providers, the communication time between clouds, the availability zone of the user, etc.

6.3. Selection of optimal cloud combination

As mentioned in Section 2, none of the existing approaches have considered the communication cost between clouds as they only focus on reducing the number of clouds. To determine the cost of communication between two clouds, it is important to consider various information, including data transfer time, pricing models, security constraints, Cloud availability zones, etc. Modeling the MCE with all of these information, using graph structure, is a natural and suitable choice, as defined below:

Definition 8 (Inter-cloud communication graph). A weighted directed graph $G = (C, A, W)$ represents the communication links between clouds. C represents the set of vertices (i.e. clouds in the MCE). A denotes the set of undirected edges connecting the vertices if, and only if, there exists a communication link transmitting information from c_x to c_y, where c_x and $c_y \in C$. For every pair (c_x, c_y) of clouds that communicate with each other, we add an arc $a(x, y)$ to A with a weight $w(x, y)$ representing the communication cost between the two clouds.

In this work, for simplicity reasons, the graph G is transformed into a matrix view (see Table 4), while a more enriched inter-cloud graph modelling including other useful information (e.g. security constraints, data transfer time, etc.) is left to the future work. For this purpose, entries of the matrix in Table 4 are simple values denoting the inter-cloud communication times (in milliseconds).

To determine the optimal cloud set, the algorithm below takes as input a set *CCS* of candidate cloud combinations and the matrix M of inter-cloud communications. The output is a cloud combination CC_{best} with a minimal size N and a minimal total cost T.

Initially, the minimal score S_{min} is set to a constant value (S_{MAX}) denoting the maximal score (line 2). For each cloud combination

Table 4
Matrix of inter-cloud communication.

	C1	C2	C3	C4	C5
C1	0	510	280	110	80
C2	510	0	320	870	180
C3	280	320	0	930	1050
C4	110	870	930	0	120
C5	80	180	1050	120	0

CC_i (line 3), the algorithm runs through the matrix of inter-cloud communication to get the set E of communication links within the ith cloud combination (line 4). This set is then used to compute the total communication cost of CC_i (see formula $\sum_{j=1}^{|E|} cost_j$ in line 5). E is the set of edges representing communication links between clouds in combination CC_i. For example, the jth edge $(1 \leq j \leq |E|)$ corresponds to the jth pair (c_x, c_y) of clouds C_x and C_y that communicate with each other. Hence, $cost_j$ in line 5 denotes the communication cost between the two clouds C_x and C_y in the jth communication link (the communication cost is represented by the weight $w(x,y)$ in Definition 8).

The score S_i of the ith cloud combination mainly depends on the number of participating clouds and the communication cost between them. Algorithm 3 calculates the score of CC_i using formula in line 5, with N_i is the number of clouds in the ith combination; $|E|$ is the number of edges representing communication links between clouds in CC_i; $cost_j$ denotes the communication cost between the two clouds in the jth communication link. The total cost for the combination CC_i is calculated by considering the sum of the communication costs in the cloud combination ($\sum_{j=1}^{|E|} cost_j$). α and β are numerical values between [0,1] which denote the importance factors of clouds' number and inter-clouds communication cost, respectively. To discourage the participation of clouds from scattered availability zones and encourage having a minimal inter-cloud communication cost as the most important objective, α value should be smaller than β (see Section 8).

If the score of CC_i is lower than the minimal score S_{min} (line 6), the current best cloud combination (CC_{best}) is updated (line 7) and the new minimal score S_i is saved (line 8). Finally, the algorithm returns the optimal cloud set CC_{best} (line 11).

Note that, in this version of work, we focused on the sequential structure of a service composition. This is why the total inter-cloud communication cost is calculated, according to the sequential execution order of services, as the sum of the communication costs in a cloud combination ($\sum_{j=1}^{|E|} cost_j$). Indeed, according to YAWL model (Gabrel et al., 2015), a composite service has four basic structures: sequential, cyclic, parallel and conditional. In the future, the proposed scoring formula will be enriched to cover all the patterns in YAWL model.

Illustration: To better understand the FCA-based approach to multi-cloud service composition, we use the example of multi-cloud lattice in Fig. 3. In this example, the multi-cloud environment consists of five clouds {C1, C2, C3, C4, C5}. Each one consists of a set of providers (with their offered services), which is a subset of {P1, P2, …, P10}.

Suppose that we have a service request $S_r = \{S1, S2, S4, S6\}$. When a user sends his request to the composition engine, there may exist many solutions in the different clouds that may satisfy his needs. These solutions are extracted based on the possible relations (within concept lattices) between providers and their offered services on the first hand, and between providers and their hosting clouds on the second hand.

By applying Algorithm 1, a set of providers that offer a part (or all) of the requested services is extracted from each Cloud lattice. Looking within the five lattices of services (L_1^C, L_2^C, L_3^C, L_4^C, and L_5^C), the candidate providers are {P1, P2, P3, P4, P6, P8, P9}. Fig. 4(a)

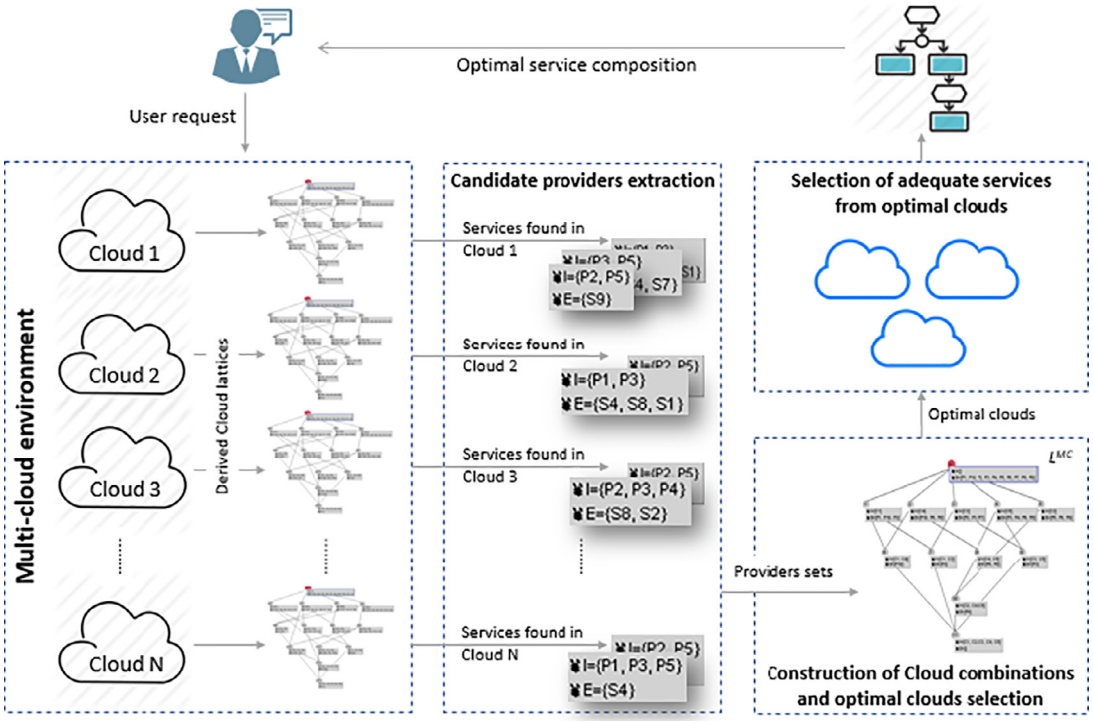

Fig. 3. Main steps of the multi-cloud service composition approach.

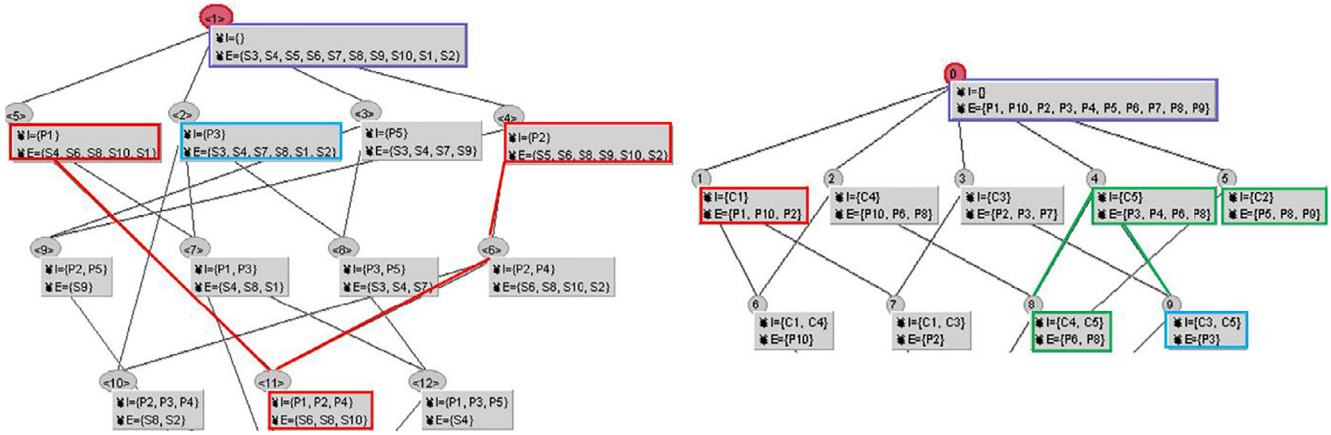

Fig. 4. Top-down lattice parsing: (a) sub-hierarchy of candidate providers' concepts in L_1^C (b) sub-hierarchy of candidate clouds' concepts in L^{MC}. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

presents a sub-hierarchy of the Cloud lattice in Fig. 2 (first three levels). This sub-hierarchy depicts the formal concepts that contain the requested services in the first cloud. Red and blue rectangles depict the pertinent concepts that offer the minimal combination of providers, whereas red lines denote the hidden associations between these formal concepts. In Fig. 4(b), the candidate clouds belong to the colored formal concepts (first two levels of the multi-cloud lattice in Fig. 1). The associations between these latter are specified using green lines. From Fig. 4(a), we can see that services S1, S2 and S4 are offered by provider P3 (see the 2nd concept in the blue rectangle) which deployed its services in two Cloud availability zones: C3 and C5 (see the 9th concept in Fig. 4(b)). Whereas the remaining service S6 is offered by providers P1 and P2 from cloud C1 (see the 1st concept in Fig. 4(b)), as well as provider P4 hosted by Cloud C5 (see the 4th concept in Fig. 4(b)).

Because we adopt a top-down approach, the formal concept with the highest size of Extent (i.e. highest number of services) is processed to confirm whether it contains a part of the requested services. In the example of Fig. 4(a), the supremum (top concept) of the lattice is ignored because the size of its Intent is equal to 0 (C1.Intent = Ø). This means that none of the providers hosted in Cloud C1 offer all of the requested services. Consequently, it is unnecessary to process this concept.

Take the example of the first cloud in Fig. 4(a). Because C2.Extent\S_r = {S1, S2, S4}, the candidate provider P3 in C2.Intent will be added to the Combiner list, and the set containing the remaining requested services is S_r = {S6}. Because S_r is not empty, the same routine is repeated using the next largest concept C4, which contains in its Extent part the rest of requested services {S6}. Consequently, candidate provider P2 in the Intent part is added to the Combiner list. The algorithm will then terminate processing the lattice of the first cloud and starts running through the next cloud lattice. At the end, a set of candidate providers from the 5 clouds will form a sub-hierarchy of the multi-cloud lattice L^{MC}.

These providers are used to extract the candidate cloud combinations according to the hidden relations between formal concepts in the multi-cloud lattice L^{MC}. For instance, providers P1 and P2 host their services in the same cloud C1, services of providers P3

Algorithm 1 Candidate providers' extraction.

Input: A user request Sr; N lattices L_1^C, L_2^C, ..., L_N^C of services and their providers.
Output: M sets of candidate service providers
Assumption: Concepts are sorted in decreasing order based of the number of offered services
01: **Begin**
02: PSs := Ø
03: **for each** cloud Cl_i **do** // perform a top-down search within each lattice
04: S_a := Ø
05: V_{prov} := Ø
06: **for each** C_j in L_i^C **do**
07: **if** $(S_r - S_a) \cap C_j$.Extent ≠ Ø **then** // check if the Extent contains services
that can fulfil user request
08: add $(S_r - S_a) \cap C_j$.Extent to S_a
09: $V_{prov} := V_{prov} \cup C_j$.Intent // add the providers to combiner list
10: **end if**
11: **if** $S_a := S_r$ **then** // user request fulfilled
12: PC := getProvidersCombinations(V_{prov})
13: PSs := PSs ∪ PC
14: Initialize S_a and V_{prov}
15: **end if**
16: **end for**
17: **end for** // check the next cloud
18: **return** PSs
19: **End**

Algorithm 2 Cloud combinations' construction.

Input: M sets PS = {PS_1, PS_2, ..., PS_M} of candidate service providers; a lattice L^{MC} of all clouds and their hosted providers.
Output: A set CCS of candidate cloud combinations.
Assumption: Concepts are sorted in decreasing order based of the number of hosted providers
01: **Begin**
02: CCS:= Ø
03: **for each** candidate providers' set PS_i in PS **do**
04: V_{comb}:= Ø
05: **for each** C_j in L^{MC} **do**
06: **if** C_j.Extent ∩ PSi ≠ Ø **then**
07: $V_{comb} := V_{comb} \cup C_j$.Intent // add the cloud to the combiner list
08: $PS_i := PS_i - C_j$.Extent
09: **end if**
10: **if** PS_i:= Ø **then** break
11: **end for**
12: CC:= getCombinations(V_{comb})
13: CCS:= CCS ∪ CC
14: **end for**
15: **return** CCS
16: **End**

Algorithm 3 Optimal cloud combination's selection.

Input: A set CCS = {CC_1, CC_2, ..., CC_m} of cloud combinations; Matrix M of inter-cloud communications.
Output: Cloud combination with minimal number of clouds and minimal communication cost.
01: **Begin**
02: $S_{min} := S_{MAX}$
03: **for each** cloud combination CC_i **do**
04: E:= getCommunicationLinks(CC_i, M)
05: $S_i := \alpha.N_i + \beta.\sum_{j=1}^{|E|} cost_j$
06: **if** $S_i < S_{min}$ **then**
07: $CC_{best} := CC_i$
08: $S_{min} := S_i$
09: **end if**
10: **end for**
11: **return** CC_{best}
12: **End**

Table 5
Candidate cloud combinations.

Providers set	Combination	Size	Cost	Score
{P1, P3}	{C1, C3}	2	560	392.6
{P1, P3}	{C1, C5}	2	240	168.6
{P2, P3}	{C3}	1	0	0.3
{P3, P4}	{C5}	1	0	0.3
{P6, P8, P9}	{C2, C5}	2	540	378.6
{P6, P8, P9}	{C2, C4}	2	870	609.6
{P1, P4, P9}	{C1, C2, C5}	3	690	483.9

and P4 are deployed in cloud C5, both providers P6 and P8 deploy their services in the same clouds {C4, C5}, whereas the remaining candidate provider P9 deploys its services on cloud C2. Thus, the possible cloud combinations for the five different providers' sets are given in Table 5.

By looking within the matrix of inter-cloud communication, Algorithm 3 can calculate the score of each cloud combination. Among these candidates, only {C3} and {C5} are the optimal cloud combinations, because they use the minimal number of clouds (only one cloud) with a minimal communication cost equal to 0 ms (see Table 5). Although the last cloud combination {C1, C2, C5} involves a higher number of participating clouds, it is considered better than the combination {C2, C4} because this later produces a higher communication cost (870 ms).

Finally, only services available on cloud C3 or cloud C5 are considered in the final step of the composition process.

6.4. Service composition selection

Once the optimal cloud combination is determined, the set of services deployed in the selected clouds forms a distributed service repository that will be used in the selection of the optimal service composition.

Hence, the last step consists of comparing the services' capabilities (functional interfaces, QoS offerings, etc.) with the requirements of each task (i.e. requested service) in the user request S_r, and then selecting a suitable service for each task. Since this is just a classical service selection problem, we use one of the existing Cloud service selection approaches to ensure service composition within the set of selected clouds. Using the selection algorithm, each task in the service query can be allocated to one service in the optimal multi-cloud repository (i.e. selected cloud combination), such that the QoS of service composition is optimized.

Note that for some cloud users, QoS is more important than communication cost, as reducing the inter-cloud communication cost does not always guarantee a high QoS of the composed service. But, it is important to notice that the service composition process is, in general, driven by the user requirements and preferences. So, there are always tradeoffs between QoS, number of combined clouds, inter-cloud communication cost, and even the cost of the service itself, when multiple clouds are considered. In this version of work, our main objective is to reduce the number of clouds, the number of providers and the inter-cloud communication cost. Our approach may be easily enhanced with QoS constraints by integrating a QoS scoring function into the proposed algorithms. Hence, low computed QoS values may be used to penalize a candidate cloud combination, even if this later guarantees a reduced number of clouds and a low communication cost.

7. Theoretical complexity study

The strong mathematical background of FCA assured an explicit representation of the multi-cloud environment. Also, the regrouping capabilities offered by the lattice theory allowed us to extract the most suitable providers and close clouds. However, finding the optimal cloud combination set comes with a price as the complexity of algorithms depends on the size of the lattices L^{MC} and L^C.

Candidate providers' extraction: The cost of this step depends mainly on the number of requested services (|Sr|) and the size

of the multi-cloud environment (N). In the worst case, the algorithm runs through the whole cloud lattice L^C to extract candidate providers. This process is repeated for each cloud, which gives us a complexity in order of $O(N.|L^C|)$. Also, Algorithm 1 checks if the Extent contains services that can fulfil a user request. The cost of this operation is $O(|Sr|^2.|C|)$, where $|C|$ is the number of services hosted by a cloud provider. The last bloc of the algorithm, which is devoted to the combination of candidate providers considers all the concepts in each lattice L^C, and each provider is combined with its peers in other Extent sections in the combinations vector V_{prov}. That is a complexity of $O(|L^C|^2)$ in the worst case. Hence, the complexity of this algorithm would be $O(Sr^2.N.|L^C|^3.|C|)$.

Cloud combinations' construction: The complexity of this algorithm depends on the set of candidate providers (PS) and the size of the multi-cloud environment. For the first input, the size of PS in the worst case is |Sr| (each requested service is offered by a different provider). For each providers' set, a total of $|L^{MC}|$ concepts will be processed to check if the Extent section of each concept contains providers that offer a sub-set of the requested services. The cost of this operation is $O(|Sr|.|PSs|^2.|L^{MC}|.|P|)$, where $|P|$ is the total number of providers in the MCE. The last bloc of the algorithm, which is devoted to the combination of candidate clouds considers all the Intent sections in the lattice L^{MC}, and each cloud is combined with its peers in other Intent sections in the Cloud combinations vector V_{comb}. That is a complexity of $O(|L^{MC}|^2)$ in the worst case. Finally, the complexity of this algorithm would be $O(|Sr|.|PSs|^2.|L^{MC}|^3.|P|)$.

Selection of optimal cloud combination: In this algorithm, we run through each cloud combinations' set (CCS) and we process the entries of inter-cloud communications matrix to calculate the total communication cost. The cost of running through the matrix is $O(N^2)$, where N is the number of clouds. The total complexity is $O(|CCS|.N^2)$.

Regarding the complexity of the proposed FCA-based method in real-word scenarios, there exists a large amount of (big) data including the high number of clouds and providers, as well as various types of services that produce a huge volume of data (e.g., conversational and transactional services, data services, and recently big services). This will generate a large formal context with overlapping relationships, but it will not have negative impacts on the complexity and the quality of our solution, because of the adopted top-down parsing method. In specific scenarios, FCA solutions may risk to suffer from a high complexity and execution time, for example, when a system needs to repetitively build the lattice from a formal context. This is the case of dynamic and real-time applications or stream-based systems, that need to rebuild the lattice to make a decision or to process the new coming/generated information. In our context, we do not need to build a lattice at each coming request, because the lattice which represents the multi-cloud repository is constructed only once at the beginning, that is, before receiving any user request.

Moreover, one of the advantages of applying FCA in our work is the ability to adopt a top-down approach, which allows parsing the minimal number of formal concepts in the lattices, and getting a minimal sub-set of services or clouds that are regrouped in the minimal number of Extent parts. Using top-down parsing, we can find the optimal combination in the first few levels of the lattice, without needing to run through the whole lattice. This has a positive impact on the processing time. So FCA can be accepted as an effective solution even in large-scale multi-cloud environment and even in case of a lower density of the multi-cloud formal context.

8. Experimental results

To verify the effectiveness of our solution to the multi-cloud service composition problem, we have implemented our FCA-based approach using Java programming language and Galicia tool. We have also developed Java classes to randomly generate some test data sets including, formal context files, data sets of hosted services, graph of multi-cloud environment. Implementation details are also available on this URL: https://goo.gl/SZuqdI. As for the simulation parameters, to determine the most relevant values of α and β, we created a simple java method that allows varying the values of α and β between 0 and 1 (knowing that $\alpha+\beta$ is always equal to 1). In this simple script, a loop is performed with 10 pairs of values $\langle\alpha, \beta\rangle$ ($\langle$0.1, 0.9$\rangle$, $\langle$0.2, 0.8$\rangle$, $\langle$0.3, 0.7$\rangle$, ..., $\langle$0.9, 0.1$\rangle$), which are used in the calculation of the cloud combinations' scores. This test is performed 10 times to see the impact of varying the values of α and β on the average produced scores. Based on this test, we came to the conclusion that α must be set to 0.3 and β to 0.7, so that to discourage the participation of clouds from scattered availability zones and encourage having a minimal inter-cloud communication cost as the most important objective.

Experiments are conducted with various numbers of clouds (between 5 and 50) and candidate services (up to 1250 services). Since the multi-cloud composition evaluation consisted of some stochastic elements in their implementation (random formal contexts, random communication costs, random binary relations, etc.), each of the test problems was repeated 10 times. The user request for all test problems contains five services.

8.1. Evaluation of computation time

The aim of the first set of experiments is to evaluate the time required to extract the candidate providers and clouds. We have varied the number of clouds between 5 and 50 clouds, with a fixed number of services within each cloud. We also have varied the density of each formal context between 20% and 40%, to see the impact of hosting providers in several clouds on the total execution time. In the experiments, we have excluded the high density values because, in real-world scenarios, a service cannot be hosted in all (or a large number of) clouds due to the financial charges and the complexity of its management. For these reasons, the density of the multi-cloud formal context should not be high (e.g., for a formal context with 10 providers and 5 clouds, if the density is equal to 80%, this means that 4 among 5 clouds, on average, host the services of each provider). Also, a very low density means that a provider hosts its services in a very few number of clouds or in only one cloud. However, we did not consider this case since our focus is on a multi-cloud setting. Based on this, we decided to vary the density between 20% and 40% for all the tests. Composition time results are given in Fig. 5.

Looking at the figure above, the execution time is slightly high when a provider is not hosted on several clouds (i.e. lower density of the multi-cloud formal context). Indeed, the algorithm, in this case, must run through the whole lattice and even may reach the infimum concept to extract the candidate clouds, which has a negative impact of the extraction time. The extraction process of cloud combinations is not affected in case of a higher density. This is explained by the ability of the algorithm to extract candidate providers and clouds from the first formal concepts using a top-down approach.

Note that even with high density of the multi-cloud formal context, the global time to extract candidate clouds and construct cloud combinations always decreases, except for the case of 50 clouds (density = 40%).

8.2. Quality of multi-cloud composition evaluation

The second set of experiments aims to evaluate the quality results of classical single-cloud service composition in comparison

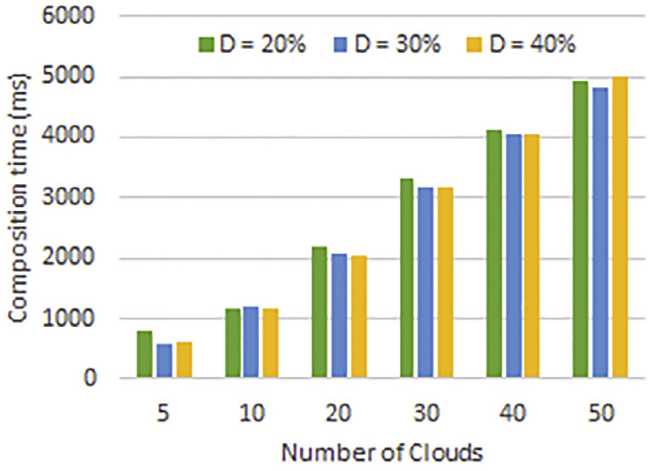

Fig. 5. Composition time with various number of clouds and various densities.

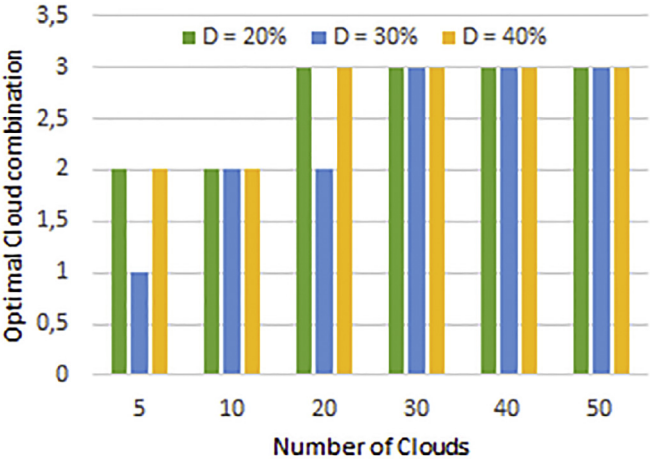

Fig. 6. Optimal cloud combinations sizes with various densities and numbers of clouds.

to the multi-cloud approach. We varied the number of clouds between 5 and 50 for different values of cloud formal context density (between 20% and 40%).

It can be seen from the above figure that the optimal cloud combination size is not affected by the variation of density of the multi-cloud formal context and when the multi-cloud space become larger (50 clouds). Moreover, the size of optimal cloud combinations has never reached the number of request services (5 services) and, in most cases, varies between 2 and 3 clouds. We also can see that the user request is satisfied by a single cloud (best case) even in a reduced density (case of 5 clouds). Fig. 6 also shows that, for cloud sizes 5 and 20, we have optimal cloud combinations for density 30% compared to 20%. This is understandable because a lower density means that a provider hosts its services in a very few number of clouds, which decreases the probability that a provider belongs to several formal concepts, because these latter will be used to determine the best cloud combination. Regarding the case of density 40% for cloud sizes 5 and 20, selecting a combination with higher size rather that a lower number of clouds, like in the case of density 30%, could be attributed to the reduced inter-cloud communication cost produced by the combination with the higher size. Although the combinations for density = 40% involve a higher number of participating clouds, they are considered better than those obtained for density = 30%, because these latter produce higher communication costs.

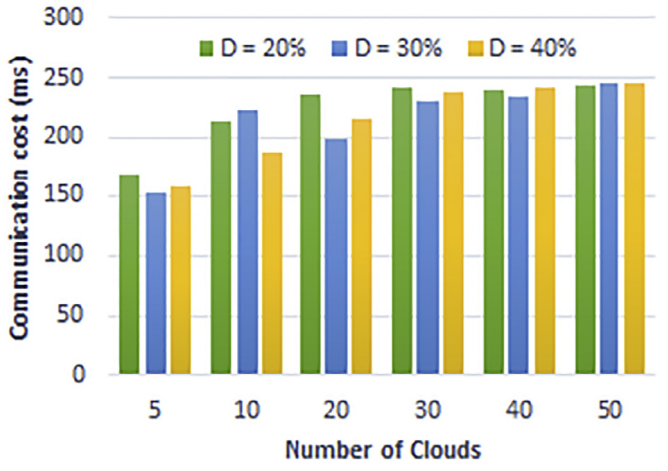

Fig. 7. Evaluation of the inter-cloud communication cost with various densities and numbers of clouds.

A higher density of the multi-cloud formal context also has a positive impact on the inter-cloud communication cost, as shown in Fig. 7. Indeed, minimizing the number of participating clouds means that at least two providers belong to the same cloud and exchange data with a low communication cost. Bars in Fig. 7 also show that the cost of inter-cloud communications is proportional to the cloud combinations size. Also, it is clear from the figure that the larger multi-cloud space has not increased the inter-cloud communication cost. In fact, cost values are between 198 ms and 244 ms in most cases, which presents a reduced cost that will not affect the total execution time of the optimal service composition.

Note that the quality of produced solution is not affected by the reduced density of the multi-cloud formal context (e.g. 20%). A reduced density means that a provider is hosted on a single cloud or at most on few clouds. This means that the size of optimal cloud combination is equal to the size of user request, in the worst case. However, one of the advantages of applying FCA is the ability to adopt a top-down approach to get a minimal sub-set of services that are regrouped in the Extent section of the lattice supremum, which means that these services are offered by the same provider. This has a positive impact on the final solution as reducing the number of providers increases the probability to find the optimal set of clouds in the Extent section of the supremum concept or in a close concept, also using the top-down approach. Experimental results showed that our FCA-based method always produces an optimal cloud combination even in large-scale multi-cloud environment and even when each provider is hosted in few numbers of clouds.

8.3. Comparison with other approaches

In this section, the performance and the solution quality of our algorithm are compared to those produced by the ant colony optimization algorithm (ACO-WSC) proposed by Yu et al. (2015) and the combinatorial optimization algorithm (COM2) proposed by Kurdi et al. (2015). To simulate the cloud environment, we used OWL-S XPlan package (Klusch et al., 2005) which offers a service test set. In OWL-S XPlan package, the MCE includes five service files $F = \{F1, F2, F3, F4, F5\}$ deployed on four clouds MCE = {C1, C2, C3, C4}. Based on these settings, we transformed these MCE data into a set of formal contexts and we created five multi-cloud formal contexts, each one with a different configuration that uses four clouds and five providers (i.e. service files). The density values of the five created formal contexts are respectively 55%, 40%, 40%, 50% and 40%. These values are determined according to the settings of-

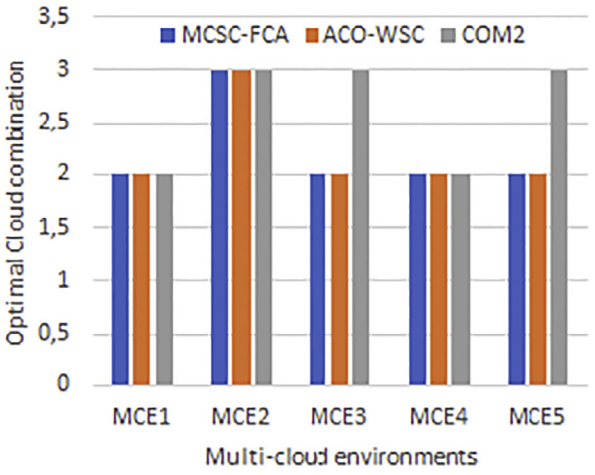

Fig. 8. Number of combined clouds in MCSC-FCA, ACO-WSC and COM2 with various MCE settings.

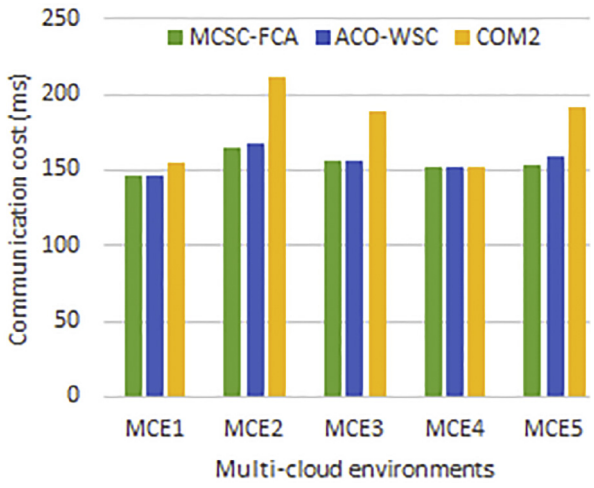

Fig. 9. Communication costs for MCSC-FCA, ACO-WSC and COM2 with various MCE settings.

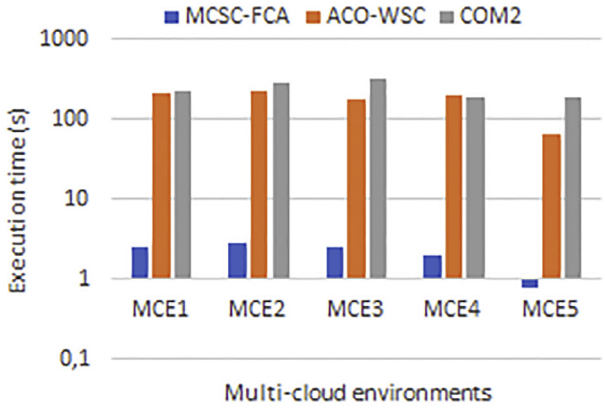

Fig. 10. Composition time for MCSC-FCA, ACO-WSC and COM2 with various MCE settings.

fered by OWL-S XPlan package, and the distribution of providers (i.e. services files) on the four clouds in each MCE. We also maintained the same simulation parameters as in Kurdi et al. (2015) to ensure a robust comparative experimental study.

Comparison of cloud combinations' sizes: Fig. 8 illustrates the sizes of optimal cloud combination for each MCE test case. Comparing the three algorithms, our method surpassed COM2 in most cases with a margin of one cloud. MCSC-FCA and ACO-WSC produced the best combinations' sizes for all MCE test cases. Additionally, the number of clouds involved in the optimal composition is always equal to the number of concepts processed in the multi-cloud lattice. These optimal results are explained by the top-down approach adopted to run through the lattice, which allowed parsing the minimal number of concepts. Even in its worst (2 clouds), the algorithm reaches the second level of the multi-cloud lattice, and always returns a few number of clouds. This low number of clouds has a positive impact on the inter-cloud communication cost.

Comparison of the inter-cloud communication costs: Since the service test set offered by OWL-S XPlan package (Klusch et al., 2005) does not include inter-cloud communication data, we created the matrix of inter-cloud communication costs with 4×4 dimension (in OWL-S XPlan package, each of the five MCEs includes 4 clouds). We randomly generated the entries of the matrix as simple values denoting the communication times (in milliseconds) between clouds. To avoid the doubt whether the random values in the matrix can influence the obtained results (total communication cost), 10 different matrices were generated for the test and used for each of the five MCE settings.

Using our defined formula (see Algorithm 3), the total cost for each cloud combination produced by MCSC-FCA, ACO-WSC and COM2 in the previous test is calculated by considering the sum of the communication costs in the cloud combination ($\sum_{j=1}^{|E|} cost_j$). The obtained results are shown in Fig. 9.

It is clear from Fig. 9 that the inter-cloud communication costs are proportional to the cloud combinations sizes obtained in the previous test. For all MCE settings, the best total cost values were obtained by MCSC-FCA and ACO-WSC. Our approach is slightly better than ACO-WSC, unlike COM2 which was influenced by the number of clouds involved in the composition.

Fig. 9 also shows that the three approaches produced the same results for some test cases. For example, MCSC-FCA and ACO-WSC produced the same communication costs for three different environments: MCE1, MCE3 and MCE4 (respectively 146 ms, 156.2 ms, and 151.6 ms). In the fourth MCE setting, the three approaches produced the same communications cost (151.6 ms). This is understandable because, as shown in Fig. 8, the approaches returned the same number of combined cloud in these MCE settings, and probably the same set of clouds in the optimal combination. However, this is not always true because the same combination size was returned for the three approaches in MCE1, but with a different communication cost produced by COM2 (154.6 ms). Same thing for the approaches MCSC-FCA and ACO-WSC in the multi-cloud setting MCE5 (respectively 153 ms and 159.5 ms). Although ACO-WSC and COM2 have produced the best combinations' sizes like MCSC-FCA, they did not return the optimal cloud set, unlike our approach. Indeed, the grouping capabilities of FCA not only retain a minimal number of clouds, but also the best cloud combination according to the possible (hidden) relations between the formal concepts that group these clouds. As for COM2, it is clear that the higher number of clouds has a negative impact on the communication cost (case of MCE2, MCE3 and MCE5).

Comparison of the execution times: Regarding the time spent to find the optimal cloud combination, Fig. 10 shows that MCSC-FCA outperforms ACO-WSC and COM2 for the five MCE test cases. This is explained by the top-down approach (adopted for optimization reasons) that allowed processing the minimal set of concepts in the lattice. Using a top-down method, we can find the optimal combination in the two first levels of the lattice, without needing to run through the whole multi-cloud lattice, and we never reach

the infimum. Moreover, the requested providers may belong to the Extent part of the supremum concept or, in the worst case, in the two first levels of the lattice.

In contrast, COM2 has to examine most available services, which has a negative impact on its execution time as shown in Fig. 10. The gap of execution time can be easily seen from the above figure. COM2 provides an optimal combination in approximately 195 s, unlike the time spent by our method which is 2.801 s in the worst case and 0.785 s in its best. Compared to COM2, ACO-WSC features a slightly better execution time. This could be attributed to the constructive greedy heuristic search offered by ACO. However, this algorithm always suffers from a high execution time because it randomly selects a cloud base during the combination step and examines a high number of services.

9. Conclusion

The aim of this work was to provide a solution to the problem of multi-cloud service composition. The main goal was to propose algorithms that could efficiently offer optimal service compositions with a short total execution time and a minimal number of clouds, thereby reducing communication costs and financial charges. The proposed method takes the advantages of the mathematical foundations of Formal Concept Analysis. We modelled the MCE in a set of concept lattices. In cloud lattices, each formal concept represents a possible cluster of services offered by the same provider(s). Whereas, in the multi-cloud lattice, each formal concept represents a possible cluster of providers that host their services within the same cloud. These clusters are then used to determine clouds combinations according to the possible (hidden) relations between two or more formal concepts. The results of our study showed that the grouping capabilities of FCA retain a high number of services offered by the same provider within the same cloud, which significantly ensures, not only a reduced number of providers, but also a minimal number of clouds. These results arise from the fact that the higher is the number of services in the Extent part of a formal concept, the lower is the number of providers and clouds.

Our experimental results also indicated that the proposed method can efficiently and effectively deliver high quality of service compositions that come from a minimum number of clouds. However, because of the complex nature of FCA method, the time devoted to explore and exploit a large number of clouds is high, although it is considerably better that those spent by ACO-WSC and COM2 methods. The future work aims to reduce the time spent to deliver a suitable service composition in a multi-cloud setting.

Also, composing services by only focusing of reducing the number of participating clouds may disclose sensitive service tasks, especially if these latter are deployed in a dynamic and an untrusted cloud environment. A challenging problem is to consider security constraints in service composition. In fact, there are several security issues when composing services with uncertain availability and security constraints. Existing secure service composition mechanisms only focus on SLA availability rates and assume a fully trusted Cloud provider, which is not always true. To address these issues, it is important to provide a service composition by combining the safest services coming from the most trusted cloud combination. Such clouds must comply at best with user's preferences and policies, and offer an appropriate level of safety. This will be the focus of our future work.

Finally, with the growing need to offer business processes as Cloud services (also called "Business Process as a Service" or BPaaS (Accorsi, 2011; Petcu and Stankovski, 2012) and to deal with the highly dynamic nature of multi-cloud environments, we look for the design and management of what we call "self-* BPaaS". The latter are Cloud-oriented business processes which have the ability to manage themselves without user intervention. Self-* service compositions are agent-oriented autonomic compositions (Chainbi et al., 2012). However, the traditional Cloud service composition only constructs the potential dependency relations according to the user's requirements among Web services. Moreover, the operation unit of traditional service composition is atomic service, and existing works seldom consider the reuse of service process fragments (SPF) in any granularity (Yang et al., 2014). Process fragments' reuse is viewed as a powerful means for the rapid construction of new services. Effectively reusing arbitrary granularities of BPaaS fragments has not been solved yet. We believe that reusing BPaaS fragments than reusing atomic Cloud services directly can not only decrease the composition time, but also improve the reliability of the whole composition process. In this context, a method for reusing BPaaS fragments based on k-cut technique is underway.

References

Accorsi, R., 2011. Business process as a service: Chances for remote auditing. In: Proceedings of 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW). IEEE, pp. 398–403.

Ardagna, D., Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., et al., 2012. MODAClouds: a model-driven approach for the design and execution of applications on multiple clouds. In: Proceedings of 2010 ICSE Workshop on Modelling in Software Engineering (MISE), pp. 50–56.

Chainbi, W., Mezni, H., Ghedira, K., 2012. AFAWS: an agent based framework for autonomic Web services. Multiagent Grid Syst. 8 (1), 45–68.

Cheng, J., Yu, J.X., Ding, B., Philip, S.Y., Wang, H., 2008. Fast graph pattern matching. In: Proceedings of IEEE 24th International Conference on Data Engineering, April 2008 (ICDE 2008). IEEE, pp. 913–922.

De Maio, C., Fenza, G., Loia, V., Orciuoli, F., 2017. Distributed online temporal Fuzzy concept analysis for stream processing in smart cities. J. Parallel Distrib. Comput in press.

De Maio, C., Fenza, G., Loia, V., Senatore, S., 2012. Hierarchical web resources retrieval by exploiting fuzzy formal concept analysis. Inf. Process. Manag. 48 (3), 399–418.

Dou, W., Zhang, X., Liu, J., Chen, J., 2015. HireSome-II: towards privacy-aware cross–cloud service composition for big data applications. IEEE Trans. Parallel Distrib. Syst. 26 (2), 455–466.

Fenza, G., Senatore, S., 2010. Friendly web services selection exploiting fuzzy formal concept analysis. Soft Comput. 14 (8), 811–819.

Ferchichi, H., Akaichi, J., 2016. Using mapreduce for efficient parallel processing of continuous K nearest neighbors in road networks. J. Softw. Syst. Dev doi:10.5171/2016.356668.

Gabrel, V., Manouvrier, M., Murat, C., 2015. Web services composition: complexity and models. Discrete Appl. Math. 196 (C), 100–114.

Gutierrez-Garcia, J.O., Sim, K.M., 2013. Agent-based cloud service composition. Appl. Intell. 38 (3), 436–464.

Hao, F., Park, D.S., Min, S.D., Park, S., 2016. Modeling a big medical data cognitive system with N-Ary formal concept analysis. In: Advanced Multimedia and Ubiquitous Engineering. Springer, Singapore, pp. 721–727.

Jrad, F., Tao, J., Brandic, I., Streit, A., 2015. SLA enactment for large-scale healthcare workflows on multi-cloud. Future Gener. Comput. Syst. 43, 135–148.

Jula, A., Sundararajan, E., Othman, Z., 2014. Cloud computing service composition: a systematic literature review. Expert Syst. Appl. 41 (8), 3809–3824.

Klusch, M., Gerber, A., Schmidt, M., 2005. Semantic web service composition planning with owls-xplan. In: Proceedings of AAAI Fall Symposium on Semantic Web and Agents, November 2005, USA, vol. 5.

Kritikos, K., Plexousakis, D., 2015. Multi-cloud application design through cloud service composition. In: Proceedings of 2015 IEEE 8th International Conference on Cloud Computing, June 2015. IEEE, pp. 686–693.

Kumar, C.A., Singh, P.K., 2014. Knowledge representation using formal concept analysis: a study on concept generation. Glob. Trends Intell. Comput. Res. Dev. 11, 306–336.

Kurdi, H., Al-Anazi, A., Campbell, C., Al Faries, A., 2015. A combinatorial optimization algorithm for multiple cloud service composition. Comput. Electr. Eng. 42, 107–113.

Kuznetsov, S.O., 2004. Machine learning and formal concept analysis. In: International Conference on Formal Concept Analysis, February 2004. Springer, Berlin, Heidelberg, pp. 287–312.

Kuznetsov, S.O., Obiedkov, S.A., 2002. Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Intell. 14 (2–3), 189–216.

Microsoft Communications & Media Industries, "Multi-cloud service delivery & end-to-end management," Ref. architecture, 2013. https://enterprise.microsoft.com/en-us/articles/industries/telecommunications/multi-cloud-service-delivery-and-end-to-end-management-reference-architecture/.

Petcu, D., 2011. Portability and interoperability between clouds: challenges and case study. In: Abramowicz, W., Llorente, I., Surridge, M., Zisman, A., Vayssière, J. (Eds.), Towards a Service-Based Internet. In: Lecture notes in Computer Science. Springer, New York, pp. 62–74.

Petcu, D., Stankovski, V., 2012. Towards cloud-enabled business process management based on patterns, rules and multiple models. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications. IEEE, pp. 454–459.
Poelmans, J., Ignatov, D.I., Kuznetsov, S.O., Dedene, G., 2013a. Formal concept analysis in knowledge processing: a survey on applications. Expert Syst. Appl. 40 (16), 6538–6560.
Poelmans, J., Kuznetsov, S.O., Ignatov, D.I., Dedene, G., 2013b. Formal concept analysis in knowledge processing: a survey on models and techniques. Expert Syst. Appl. 40 (16), 6601–6623.
Qi, L., Dou, W., Zhang, X., Chen, J., 2012. A QoS-aware composition method supporting cross-platform service invocation in cloud environment. J. Comput. Syst. Sci. 78 (5), 1316–1329.
Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., Xu, X., 2014. Web services composition: a decade's overview. Inf. Sci. 280, 218–238.
Venkat, M., 2016. Enterprise cloud strategy: Governance in a multi-cloud environment. IBM. https://www.ibm.com/blogs/cloud-computing/2016/11/enterprise-governance-multi-cloud/.
Wei, Y., Blake, M.B., 2010. Service-oriented computing and cloud computing: challenges and opportunities. IEEE Internet Comput. 14 (6), 72.
Weng, S.S., Tsai, H.J., Liu, S.C., Hsu, C.H., 2006. Ontology construction for information classification. Expert Syst. Appl. 31 (1), 1–12.
Wu, T., Dou, W., Hu, C., Chen, J., 2017. Service mining for trusted service composition in cross-cloud environment. IEEE Syst. J. 11 (1), 283–294.
Yang, R., Li, B., Wang, J., He, L., Cui, X., 2014. SCKY: A method for reusing service process fragments. In: Proceedings of 2014 IEEE International Conference on Web Services (ICWS), June 2014. IEEE, pp. 209–216.
Yu, Q., Chen, L., Li, B., 2015. Ant colony optimization applied to web service compositions in cloud computing. Comput. Electr. Eng. 41, 18–27.
Zhang, F., Hwang, K., Khan, S.U., Malluhi, Q.M., 2016. Skyline discovery and composition of multi-cloud mashup services. IEEE Trans. Serv. Comput. 9 (1), 72–83.
Zou, G., Chen, Y., Xiang, Y., Huang, R., Xu, Y., 2010. AI planning and combinatorial optimization for Web service composition in cloud computing. In: Proceedings of CCV Conference, May 17–18, 2010, Singapore.

Haithem Mezni received the Ph.D. degree in Computer science from Manouba University in 2014. He is currently an Assistant Professor at Jendouba University and a member of SMART Research Laboratory, Tunisia. His current research interests include Service lifecycle management, Cloud computing and Big data. Haithem Mezni has also supervised several research works on Web service composition and adaptation, Cloud service provisioning and recommendation, Cloud resource management, etc. His publication record includes articles in peer-reviewed journals.

Mokhtar Sellami is a Technologist in computer science at the High Institute of Technological Studies in Jendouba, Tunisia. He received the M.S. degree from Jendouba University, in 2005 and the Ph.D. degree in computer science from Tunis El-manar University in 2017. Currently, he is a Member of RIADI research laboratory. His research areas include information security, databases security, and Big data.