Journal of Network and Computer Applications 39 (2014) 214–222

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

A self-organizing P2P framework for collective service discovery ☆

Carlo Mastroianni *, Giuseppe Papuzzo

ICAR-CNR, via P. Bucci 41C, 87036 Rende (CS), Italy

ARTICLE INFO

Article history:
Received 8 January 2013
Accepted 15 July 2013
Available online 1 August 2013

Keywords:
Bio-inspired
Peer-to-peer
Resource discovery
Self-organization
Service composition

ABSTRACT

With the advent of the "Internet of Services" and the use of novel technologies such as Software as a Service and Cloud Computing, the market of Information Technology is experiencing an important shift from the request/provisioning of products toward a scenario where everything, computing, storage, applications, is provided as a network-enabled service. When no single service is able to address the requirements of a specific problem, the solution can be the deployment of a workflow composed of several basic services. This paper presents a P2P self-organizing framework that facilitates collective discovery requests, issued to locate all the components of a workflow. The information system is organized so that the descriptors of services that are often used together in the same workflow are placed close to each other. This allows queries to rapidly find several target services, which helps to lower the search time and reduce the computing and network load.

1. Introduction

The "Internet of Services" is rapidly emerging as a paradigm that allows users and companies to access computational resources (storage, applications, infrastructures) as network-enabled services provided on pay-as-you go basis (European Community for Software & Software Services, 2009). The Internet of Services is expected to have a great impact not only in business scenarios, but also in advancing public administration procedures and services. The shift from the request/provisioning of products to the service-oriented view is also favored by the Cloud technology, which is making on-demand computing a common practice for many enterprises and scientific communities (Buyya et al., 2009).

In this context, one of the main requirements is to deploy a scalable architecture where services provided by different vendors or institutions can be accessed in a uniform fashion and integrated. For example, strategies are being devised to integrate the services offered by multiple Cloud companies, in scenarios referred to as Hybrid Cloud Computing or Sky Computing (Riteau et al., 2010). This trend toward integration comes from many reasons: one is the quest for elastic solutions, as the presence of multiple providers can help to adapt the amount of available resources to changing user requirements. A second reason is the necessity of avoiding vendor lock-in situations, as customers appreciate the possibility of easily moving their data and applications from one vendor to another, depending on the experienced cost and quality of service. Perhaps the most relevant motivation for this trend is that a solution cannot always be offered by a single service, but through the composition of multiple basic services in a workflow (Dustdar and Papazoglou, 2008; Srivastava and Koehler, 2003). Unfortunately, as the number of available services increases, the number of composition possibilities grows exponentially. This is the problem of combinatorial explosion (European Community for Software & Software Services, 2009).

A great research effort is currently devoted to the building of automatic and semi-automatic frameworks and tools that assist the user in three main tasks: the design of complex services and workflows; the discovery of the basic component services; and the execution of the workflows. The design phase is often assisted by tools that exploit statistics on the way services have been selected and composed in the past (Wisner, 2006), and may use semantic and ontology-oriented algorithms. Once a composite service has been designed, the basic services specified in the composition pattern must be discovered on the network. In most cases, the user does not need to discover a specific service, but a set of services having some desired characteristics, among which the most convenient ones will be selected at execution time. Discovery requests must be issued to find *service descriptors*, which contain information about the services' characteristics and the modalities to access them. In the case of a composite service, a request is needed for each component/basic service, which can result in long overall discovery times and high network loads.

This paper presents an approach that helps to improve the efficiency of "collective" discovery requests. The main idea is to spatially cluster service descriptors on the basis of the *co-use frequencies* of corresponding services. In other words, the descriptors

☆This paper is an extended and revised version of the paper (Forestiero et al., 2010b).

* Corresponding author. Tel.: +39 328 0476390; fax: +39 0984 839054.
E-mail addresses: carlo.mastroianni@gmail.com, mastroianni@icar.cnr.it (C. Mastroianni), papuzzo@icar.cnr.it (G. Papuzzo).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.07.002

of services that are often executed in the same workflow are placed in restricted regions of the network. This allows search messages to find all or most of the required basic services on a small number of close-by hosts, so as to reduce the response time experienced by the user and save processing load and bandwidth.

The presented framework is decentralized and self-organizing, which helps to endure good scalability and fault tolerance characteristics. Such properties emerge from simple operations performed by a set of mobile agents, whose behavior is partially inspired by *ant algorithms* Bonabeau et al. (1999). In numerous scientific projects, ant-inspired agents are used to reorganize items in a distributed environment (resembling the clustering of larvas performed by some species of ants Handl and Meyer, 2007), or to find a good path toward a target resource, for example a food source (Doerner et al., 2009). Both these features are exploited in the presented system by two types of agents: *ant agents* and *query agents*. Ant agents travel the network exploiting peer-to-peer (P2P) interconnections among hosts, and relocate service descriptors through probabilistic *pick* and *drop* operations. Query agents exploit the reorganization of descriptors to drive discovery messages, hop by hop, towards the target services specified in the collective discovery requests. Simulation-based analysis shows that the ant agents succeed in the spatial ordering of descriptors, and query agents are able to satisfy user requirements rapidly and with low resource consumption.

The rest of the paper is organized as follows: after the discussion of the state-of-the-art in Section 2, Section 3 describes the algorithms for the reorganization and discovery of service descriptors, Section 4 presents performance evaluation results, and Section 5 concludes the paper.

2. Related work

The development of applications over complex and distributed environments, such as Grids and federated Clouds, requires the ability to select and integrate inter-organizational and heterogeneous services. If no single service can satisfy the functionalities required by the user, it is often possible to combine existing services in order to fulfill the request. This trend has triggered a considerable number of research efforts, in academia and industry (Rao and Su, 2004; Papazoglou et al., 2007; Zimeo et al., 2008), on the composition of services. Despite these efforts, this is still a highly complex task, mainly for two reasons: (i) the number of available services increases dramatically and composition alternatives grow exponentially and (ii) services are often provided by different organizations, which use different concept models for their descriptions, and yet there is no unique language to define, evaluate and access services.

Today service composition is typically performed through centralized middleware components, such as registries, discovery engines and brokers, but this approach is inevitably a serious bottleneck and is not well scalable. In the last few years, distributed registries have been proposed to let service networks scale to the growing number of service providers and consumers. Meteor-S (Verma et al., 2005) and Pyramid-S (Pilioura et al., 2004) propose scalable P2P infrastructures to federate the registries that publish and discover services. These systems use an ontological approach to organize registries: semantic classification is based on the specific application domain and is assisted by domain experts.

Most P2P systems adopted today are *structured*, which means that peers are organized in a predefined overlay (e.g., a ring or a multi-dimensional space). A hash function is defined to assign each resource/service an access key and each peer a code, and every service is assigned to the peer whose code matches the resource key. This approach ensures that a single service can be found rapidly through an *informed* discovery procedure. Unfortunately, it is recognized (Cheema et al., 2005) that the hash function does not preserve the semantic features of services, and the keys of services that are in some way related to each other are inevitably dispersed to distant regions of the network. This means that the target services of multiple discovery requests must be found with distinct search procedures, with no chance of using the results of one request to speed up or facilitate another request.

Recently, we presented agent-based algorithms for P2P systems that do not use hash functions to compute resource keys, but assign similar keys to similar resources (Forestiero et al., 2008a, 2010a). This helps to serve *range queries*, i.e., queries for which some features of the target resources are defined through ranges of contiguous values. However, services that are frequently used in the same workflow are not *similar* to each other, as they are generally used for different purposes. For example, a preprocessing service is not similar to a data mining service, yet they are often executed in the same workflow sequence. The approach of this paper is to use the statistics on the execution of workflows to define a peculiar type of similarity between services, which corresponds to the frequency with which they are used together. This allows to define self-organizing algorithms that cluster frequently co-used services and in this way assist collective service requests. To the best of our knowledge this is the first time that an approach of this kind is proposed. In general, however, the collective discovery and composition of services can be favored by intelligent mechanisms that use semantic information about services, or that exploit statistical information about the way they have been used in the past. Some works that follow such avenues are summarized in the following.

The system presented in Birukou et al. (2007) aims to improve Web service discovery. Data collected from past requests, and corresponding service invocations and executions, are used to compute similarity between user requests, and similarity information is used to recommend useful services. Magallanes (Rios et al., 2009) is a platform-independent Java library of algorithms aimed at discovering bioinformatics web services and associated data types. Magallanes adopts a scoring system based on the number of occurrences and relative positions of matching hits, and is currently endowed with AND/OR operators and regular expressions. In Rasch et al. (2011), a set of algorithms is proposed to continuously present the most relevant services to the user, trying to anticipate user implicit requests on the basis of the current context.

The strategy of BioMoby (Di Bernardo et al., 2008) is to simplify interactive service composition for the bioinformatics domain. In each step of the workflow construction process, only those services that are compatible and more likely to be useful are displayed. This is achieved by ranking the services according to several aspects, such as semantic similarity of data type inputs or statistics about past requests. The system presented in Sellami et al. (2009) exploits past users characterizations to route a query to the most appropriate registries, exploiting P2P interconnections among registries on top of a JXTA platform. When no formal description of the composition process is easily obtained, or when the quality of existing documentation is uncertain, the use of process mining techniques can be a valid solution (van der Aalst et al., 2007a). Process mining aims at the automatic building of composite services starting from the behavior deducible from execution logs of basic services.

In Meyer et al. (2006), the authors present an approach to schedule the execution of composite services on the basis of the spatial proximity of files and jobs. The idea is to leave the files at the sites where they have been processed, so that adjacent jobs can immediately retrieve the files and remove them only when they are no longer needed. A comparison between this approach

and the one adopted here may be interesting: in Meyer et al. (2006) the location of services and resources is the input, and workflows are scheduled so as to optimize execution efficiency in the given scenario; with our approach, service descriptors are *relocated* so as to improve the performance of discovery requests.

An interesting approach is proposed in Maamar et al. (2011). The authors look into the possible overlap between social computing and service-oriented computing. The objective is to let users capitalize on their past interactions. For example, users can obtain information from Web services about the other users with whom they could profitably collaborate to build new complex services.

3. Ant-inspired algorithms for collective service discovery

As discussed in the introductory section, one of the main tasks involved in the process of building a composite service, or "workflow", is the discovery of the basic component services over the network. This paper presents a technique that facilitates and speeds up this important task.

In the examined scenario, services are provided by a large number of hosts in a distributed environment, such as a P2P system, a Grid or a Cloud. The technique can be adopted whenever a few assumptions apply, as explained in the following. The first assumption is that services can be categorized into a number of *service classes*, where this classification is driven by service semantics and functionalities. Indeed, very often users are not interested in using a specific service, but wish to collect information about a number of services having specified characteristics or functionalities, for example mathematical services that provide a numerical solution to differential equations, or bio-informatic modules that perform particular operations on protein data. Once several services have been discovered for a given class, the most appropriate one may be chosen on the basis of user-defined criteria: cost, effectiveness, availability of the service provider, etc.

The second assumption is that workflows are built following recurrent patterns (Meng et al., 2008). For example, the execution of a data mining software often requires a specific algorithm to preprocess input data: in this case, the preprocessing algorithm and the data mining software will be executed together and will often be included in the same collective discovery request.

The third assumption is that the services belonging to a given class are associated with a specific value of a *key*. This strategy is very common in P2P systems and service-oriented architectures (Taylor, 2004). A service key may be generated by a hash function, as in *Distributed Hash Tables*, or may have a semantic meaning: for example, if each bit in the key corresponds to a specific topic, the 1/0 value means that the service covers/does not cover that topic.

The fourth assumption is that a user, before using a service, must first retrieve a metadata document, or *descriptor*, which contains the reference to the service along with the description of its functionalities. The service is provided by a specific host, but the descriptor can be relocated on another host, so as to facilitate the service discovery process. This is very frequent in distributed systems, which mainly differ for the strategy adopted to relocate descriptors. For example, in most P2P systems the descriptor is maintained by a peer whose index is the same as the resource key, or is very similar to it, whereas in gossip-based systems descriptors are disseminated through the connections between adjacent peers.

Under the mentioned assumptions, a user who wishes to build a composite service must first individuate the classes of the component basic services (and the corresponding keys), then issue a search request for each class, and finally select among the services that have been discovered. This process may result in long search times and excessive use of network and computing facilities, especially if the number of component services is large. The goal of the technique presented here is to reduce the duration and load of the process by spatially clustering the descriptors of services that are frequently co-used in the same workflow. This ensures that once a discovery procedure has found some target services, with high probability it will find many other useful services in the same region of the network, and often in the same peer.

System operations are executed by three distributed algorithms: Alg. 1, used to collect and disseminate statistics about the co-occurrence of service classes in the same workflow; Alg. 2, used to relocate and cluster the descriptors of frequently co-used services[1]; Alg. 3, used to drive discovery requests toward target keys.

The algorithms are executed by mobile agents whose behavior is partially inspired by ant colonies (Bonabeau et al., 1999; Forestiero et al., 2008b). Specifically, two different types of mobile agents are defined: *ant agents* and *query agents*. Ant agents execute Alg. 2: they move and cluster the keys over the network on the basis of co-use statistics. Query agents execute Alg. 3: they are generated by users to serve collective discovery requests, and move toward the target services exploiting P2P connections. Finally, ant agents and query agents cooperate to execute Alg. 1: the former carry statistical data from peer to peer, helping the peers to make these statistics more consistent and stable, while query agent update the co-use statistics maintained by the peers through which they pass.

Figure 1 summarizes the tasks performed by agents on a single peer. The upper part and lower part of the figure show the tasks executed by ant agents and query agents, respectively, while the middle part shows the information maintained by the peer, and used/updated by agents: a repository of keys, a list of adjacent peers, and the *co-use matrix*, which collects the co-use statistical data. When a query agent arrives at the peer carrying a collective query, (i) it uses the query itself to update the co-use matrix, then (ii) it collects the target keys found in the local repository, and finally (iii) it selects the adjacent peer that most probably stores some useful keys and hops there. When an ant agent arrives at the peer, it carries the co-use matrix of the last visited peer and possibly some keys picked on the peers visited previously. The ant operates as follows: (i) it merges the carried co-use matrix with the matrix stored locally, using the *Push-Sum* protocol (Kempe et al., 2003); (ii) it executes probabilistic trials, in order to decide whether or not to drop the carried keys and/or pick other keys stored in this peer; and (iii) it hops to a new peer, chosen randomly. Ant and query agents operate continuously and in parallel. This allows the reorganization of descriptors to adapt dynamically to user requests, so as to serve them more rapidly and efficiently.

The three algorithms are described in the following subsections in detail. To help the description, it is assumed that services are composed according to the workflow pattern depicted in Fig. 2, which corresponds – with a few simplifications – to "*MassBank to KEGG*", a workflow schema for the analysis of bioinformatics data, published at the repository *myExperiment*.[2] *myExperiment* is a collaborative environment where scientists publish and share their workflows and experiment plans, and is currently the largest public repository of scientific workflows. The mentioned workflow pattern is used to retrieve information from several databases about organic molecules and their interactions, parse this information and put it into a standard format for successive processing. The workflow pattern includes 20 service classes and has two OR ramifications: a workflow instance may follow any of the branches

[1] Since each service class is associated with a specific value of the descriptor key, the discussion will often refer to the relocation and clustering of keys, for the sake of simplicity.

[2] http://www.myexperiment.org/workflows/741.html.

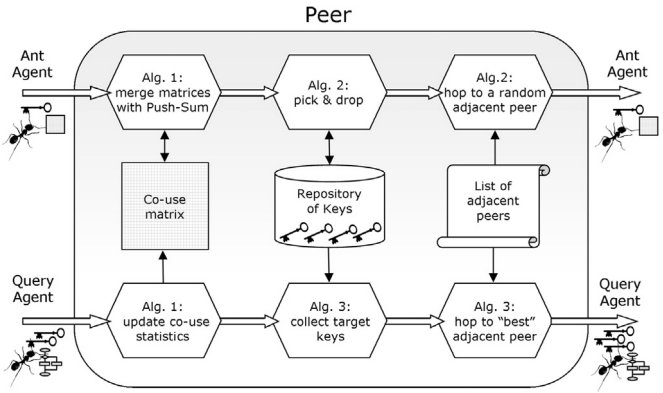

Fig. 1. Tasks performed by ant and query agents when arriving at a new peer. Each task is associated with one the three basic algorithms.

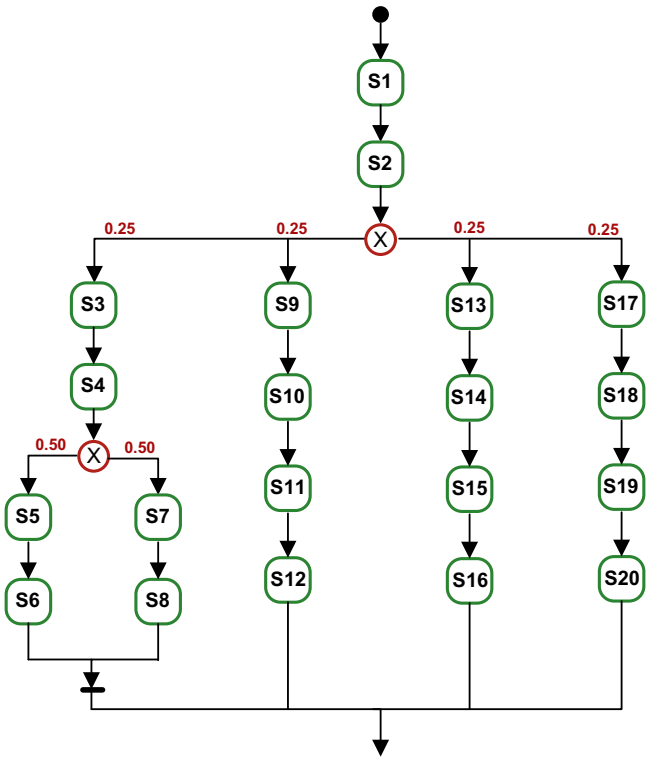

Fig. 2. Pattern of the bioinformatics workflow pattern *MassBank to KEGG*, published by the *myExperiment* repository.

of a ramification and execute the services of the chosen branch. The first two services, denoted as S1 and S2, are included in any workflow instance and are used, respectively, to select the pertinent database and prepare the query in a common format. Then, the workflow instance selects one of the branches of the first OR ramification depending on the specific database that needs to be interrogated. For example, if the workflow instance needs to download data from the PUG database (PubChem Power User Gateway, at http://pubchem.ncbi.nlm.nih.gov/), it follows the leftmost branch, which includes services from S3 to S8. Conversely, if the database of interest is ChemSpider (http://www.chemspider.com), the instance follows the rightmost branch, with services from S17 to S20. As data about the execution of workflow instances is not available, in our experiments it is assumed that all the branches are chosen with equal probabilities, as indicated in the figure.

In general, the workflow pattern makes it evident that some services are always executed together while other services are never executed in the same workflow. It is then reasonable that a collective request may be highly facilitated by the fact that the component services of a workflow are located close to each other.

3.1. Alg. 1: building and dissemination of co-use matrices

The objective of Alg. 1 is the building of *co-use matrices* on the peers of the network. Assuming that services are categorized into N_S classes, in accordance with a domain specific classification, these classes are assigned progressive integer keys from 0 to N_S-1. Each peer maintains a bi-dimensional matrix M, in which the element $M(i,j)$, with $i,j=0..N_S-1, i\neq j$, represents the frequency of co-occurrences of service classes i and j in the same workflow.

To build M, each peer examines the target keys of the collective discovery requests carried by *query agents*. For each couple of keys, i and j, the matrix element $M(i,j)$ is incremented by 1. To give more relevance to the recent behavior of the system, matrix elements are computed in a temporal window that includes the workflow requests received during the past H hours (H is a tunable parameter, and in this work it is set to 24 h). Values of M are normalized, so as to obtain values between 0 and 1.

The *ant agents* are assigned the responsibility of disseminating the values of co-use matrices among the peers, in order to speed up the convergence process. After hopping from one peer to another, an ant delivers the matrix of the source peer, M_s, to the target peer. The target peer updates its co-use matrix M_t by combining its current values with the values of the matrix M_s. This technique follows the *Push-Sum* protocol (Kempe et al., 2003), which allows aggregate values, in this case the average values of the M elements stored by different peers, to be computed in a number of rounds that is logarithmic with respect to the number of peers. In this way, information is progressively disseminated across the network, and the values of the peer matrices converge rapidly to consistent values.

Figure 3 shows the expected values of the co-use matrix when services are composed following the workflow pattern of Fig. 2. Service classes are grouped in blocks, since services belonging to the same block are always executed together. The matrix is symmetrical and sparse, hence only a fraction of its values must be stored by peers and carried by ant agents.

	S1-S2	S3-S4	S5-S6	S7-S8	S9-S12	S13-S16	S17-S20
S1-S2	1.00	0.25	0.125	0.125	0.25	0.25	0.25
S3-S4	0.25	0.25	0.125	0.125	0.00	0.00	0.00
S5-S6	0.125	0.125	0.125	0.00	0.00	0.00	0.00
S7-S8	0.125	0.125	0.00	0.125	0.00	0.00	0.00
S9-S12	0.25	0.00	0.00	0.00	0.25	0.00	0.00
S13-S16	0.25	0.00	0.00	0.00	0.00	0.25	0.00
S17-S20	0.25	0.00	0.00	0.00	0.00	0.00	0.25

Fig. 3. Co-use matrix for the myExperiment workflow pattern. Service classes are grouped in blocks, and values are equal within each block.

3.2. Alg. 2: reorganization of descriptors

The objective of this algorithm, executed by the ant agents, is the reorganization of keys. An ant is generated by a peer when it joins or reconnects to the system, and travels the network hopping from peer to peer. The lifetime of an ant is set to the average connection time of the issuing peer. In this way, the average number of circulating ants, $\overline{N}_a$, is always comparable to the average number of peers connected to the network, $\overline{N}_p$, since dying ants are compensated by new ants generated by reconnecting peers.

Each ant agent, at random time intervals, hops from a peer to an adjacent one, chosen randomly. At the new peer, the ant tries to *pick* a key that is not frequently co-used with the others stored locally. After picking a key, the ant carries it and tries to *drop* it in a peer where the co-use frequency with the local keys is high. The continuous execution of pick and drop operations ensures that the descriptors of frequently co-used services are kept close to each other. Pick and drop operations are described in the following.

Pick operation. To decide whether or not to pick a key from a peer, say key $\hat{k}$, the ant agent computes the "co-use similarity" of the key with the other keys k stored in the local area A, which includes the local peer and the peers directly connected to it. The co-use similarity $f(\hat{k},A)$ is defined as follows:

$$f\left(\hat{k},A\right)=\frac{1}{N_k}\cdot\sum_{k\epsilon A}\left(M\left(k,\hat{k}\right)\right) \tag{1}$$

where M is the co-use matrix of the local peer, and N_k is the number of keys stored in the local area A, excluding $\hat{k}$. The co-use similarity has values between 0 and 1, and a higher similarity corresponds to a higher co-use frequency. The agent performs a Bernoulli trial, i.e., a trial with two possible outcomes, success (the key is picked) or failure (the key is left at the peer). The success probability is

$$P_{pick}=\frac{\alpha_p}{\alpha_p+f(\hat{k},A)},\quad \text{with } 0\le\alpha_p\le 1 \tag{2}$$

Therefore, the pick probability is inversely proportional to the co-use similarity, which ensures that an "outlier" key will be picked with high probability and moved to other regions of the network. The parameter α_p can be tuned to modulate the degree of similarity among keys. In this work, α_p is set to 0.1, as in the canonical ant algorithm (Bonabeau et al., 1999).

With reference to the co-use matrix shown in Fig. 3, if all the keys stored in the local area belong to service classes S3–S8, a key of class S10 will be picked with high probability, because its similarity with the other keys is zero. This is also evident in Fig. 2: a service of class S10 is never executed in a workflow with any of the services S3–S8.

Drop operation. After a key $\hat{k}$ has been picked, it is carried by the ant agent across the network. The agent evaluates the drop probability function at any new peer, until the key is dropped; then, the agent will try to pick another key. The drop operation is also based on a Bernoulli trial, whose probability is

$$P_{drop}=\frac{f(\hat{k},A)}{\alpha_d+f(\hat{k},A)},\quad \text{with } 0\le\alpha_d\le 1 \tag{3}$$

This time the drop probability is directly proportional to the co-use similarity $f(\hat{k},A)$ between the carried key and keys stored locally. The value of α_d is set to 0.5. For example, with reference to the workflow pattern in Fig. 2 and the co-use matrix of Fig. 3, a key S10 is dropped with high probability if the local keys belong to classes S9–S12, while it is not dropped if local keys belong to classes S3–S8.

Being concurrently executed by many ant agents, pick and drop operations cluster keys of service classes having high co-use frequencies, and avert keys of rarely co-used service classes. The efficiency of this technique lies in the "swarm intelligence" behavior of ant-inspired algorithms.

3.3. Alg. 3: collective discovery of services

The purpose of this algorithm is to find as many services as possible that belong to a set of target classes and are needed to build and execute a workflow instance. The reorganization of keys achieved with Alg. 2 is exploited to increase the efficiency of the discovery process. Specifically, the process exploits the fact that keys are clustered (equal or similar keys are often stored in the same peer) and spatially sorted (close-by peers store similar keys). These aspects will be quantitatively examined in Section 4.1.

When a user initiates a discovery process, it issues a collective query and consigns the list of target key values (each corresponding to a service class) to a *query agent*. The query agent follows a path toward the target keys exploiting their spatial sorting. It operates as follows:

(i) first, it selects the service class for which it is most likely to find a good number of keys in the local area. To do this, the agent computes the similarity of each key of the collective query with the keys stored in the local area, using expression (1), and selects the key k_x that maximizes this function. Now the objective is to find as many keys k_x as possible.
(ii) the agent hops to the neighbor peer at which the similarity of key k_x with the keys stored locally is the largest. This step is repeated until the similarity computed at the current peer is larger the similarity computed at any of the adjacent peers. This means that the current peer most probably maintains the largest number of keys k_x. At this point, the query agent collects the keys k_x discovered so far, and removes k_x from the list of keys that are not yet selected.
(iii) the agent turns to step (i) and selects, among the keys contained in the collective query and not yet selected, the key k_y for which the similarity with the keys stored locally is the largest. The agent executes step (ii) to discover keys with the new target value k_y.

The procedure terminates when all the keys of the collective query have been selected. During the search path, the query agent collects all the target keys found along the path. At the end of the procedure, the query agent goes back to the requesting peer, which examines the keys discovered for each service class. If the result is satisfactory, the discovery process terminates. Otherwise, a new query agent is issued to search for the service classes for which the required number of keys have not yet been discovered. The whole process terminates when the required number of keys have been found for each service class, or when it is not possible to discover more keys.

As an example, let us assume that the workflow to be executed adheres to the pattern of Fig. 2 and follows the branch that includes service classes S1, S2, S9, S10, S11 and S12. A query agent is then issued to discover keys of these services. Let us further assume that the required number of keys per class is set to 10, and that the query agent returns with 12 keys for each of the service classes S1, S2 and S10, and 6 keys for each of the service classes S9, S11 and S12. The request is satisfied for the first three classes, but it is not for other three. Therefore, a second query agent is issued to find 4 more keys for classes S9, S11 and S12. If this second agent succeeds, the whole request is satisfied using only two query agents.

Notice that, in absence of key reorganization, six query agents or messages would be needed, one for each service class specified in the workflow. Owing to the described reorganization of keys,

	a	b	c	d	e	f	g	h	i	j	k	l
a	1.00	0.89	0.90	0.90	0.90	0.89	1.00	0.11	0.09	0.09	0.11	0.02
b	0.89	0.90	0.90	0.90	0.89	0.89	0.89	0.00	0.00	0.00	0.00	0.00
c	0.90	0.90	0.92	0.92	0.92	0.91	0.91	0.00	0.00	0.00	0.00	0.00
d	0.90	0.90	0.92	0.92	0.92	0.91	0.91	0.00	0.00	0.00	0.00	0.00
e	0.90	0.89	0.92	0.92	0.92	0.91	0.91	0.00	0.00	0.00	0.00	0.00
f	0.89	0.89	0.91	0.91	0.91	0.91	0.91	0.00	0.00	0.00	0.00	0.00
g	1.00	0.89	0.91	0.91	0.91	0.91	1.00	0.11	0.09	0.09	0.11	0.02
h	0.11	0.00	0.00	0.00	0.00	0.00	0.11	0.11	0.09	0.09	0.11	0.02
i	0.09	0.00	0.00	0.00	0.00	0.00	0.09	0.09	0.09	0.09	0.09	0.00
j	0.09	0.00	0.00	0.00	0.00	0.00	0.09	0.09	0.09	0.09	0.09	0.00
k	0.11	0.00	0.00	0.00	0.00	0.00	0.11	0.11	0.09	0.09	0.11	0.02
l	0.02	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.00	0.00	0.02	0.02

Fig. 4. Co-use matrix resulting from the execution of ProM workflow instances. Service classes are labeled with letters from *a* to *l*.

it is possible to reduce the number of query messages, down to only one in most cases. This aspect will be examined in Section 4.2.

4. Performance evaluation

The performance of the presented framework has been evaluated for two sets of experiments. For the first set we used the myExperiment workflow pattern shown in Fig. 2, and generated 1000 workflow instances that follow the pattern. For each workflow instance, a collective query is issued to search for the services that compose the instance. The co-use matrix for these experiments was reported in Fig. 3.

For the second set of experiments we used a set of workflow logs published at the *ProM* Web site.[3] ProM (van der Aalst et al., 2007b) is an open-source framework aimed at implementing and assessing process mining tools in a standard environment. ProM workflow logs are widely used in the literature to test algorithms for mining process models, and their publication on the Internet guarantees the full reproducibility of the results. Specifically, we reproduced the workflow instances reported in the log file `a12f9n10_5.xml`, which contains 1800 workflow traces. Services are categorized into 12 classes, and the average number of services per trace is 6.72. As for the first set of experiments, for each workflow trace a collective discovery request is issued to search for the corresponding component services. The co-use matrix of ProM experiments is reported in Fig. 4. The large variability of matrix values testifies that some services are co-used very frequently (for example services belonging to classes *a* to *g*), while others are hardly used in the same workflow.

The simulation experiments were performed with an event-based simulator similar to that used in Forestiero et al. (2008b): the simulation evolves through messages exchanged among objects, i.e., among peers and mobile agents. Performance indices were evaluated for a network of 2500 peers connected in a scale-free topology, in which the number of connections of a peer follows the power-law distribution (Barabási and Albert, 1999), and the average is set to 4 neighbors per peer. Each peer publishes 15 services on average, with the actual number extracted from a Gamma statistic distribution. The class of each published service is generated randomly, with a uniform distribution. Collective discovery requests are generated at the rate of one request per peer every 1000 s.

Peers are not stable, but connect and disconnect periodically. The connection time of the peers is extracted from a Gamma distribution with average set to 4 h. Each reconnecting peer generates an ant agent, whose lifetime is set to the connection time. This ensures that the overall number of circulating agents is kept approximately constant, and is comparable to the number of connected peers, since dying agents are substituted by other agents generated by the peers that enter the network.

The performance of the framework was evaluated with regard to three aspects: the capacity of reorganizing and clustering the descriptor keys, the effectiveness and efficiency of discovery operations, and the load induced by ant and query agents. These aspects are examined in the following subsections.

4.1. Performance of the key reorganization process

The objective of the key reorganization process is to put the keys of co-used services in the same or in neighbor peers. To this aim, two indices are defined: the peer *homogeneity*, which assesses the co-use frequency of the keys stored in a single peer, and the peer *similarity*, which evaluates the co-use frequency of the keys stored in a local region of the network that includes a peer and its neighbors (the neighbors of a peer are the peers directly connected to it). Both indices should be as high as possible: a high value of homogeneity means that a discovery request can find many useful keys in the same peer; a high value of similarity is a sign that the keys of neighbor peers are similar, and therefore keys are spatially sorted on the overlay with respect to their co-use frequency. This also means that the target keys of a collective request can be found in a restricted region of the network, with few steps of query agents.

More specifically, the *homogeneity* of a peer P is defined as the average value of the co-use frequency, evaluated for every couple of keys stored in P:

$$O_m(P) = \frac{1}{N_{k_1,k_2}} \cdot \sum_{k_1,k_2 in P} M(k_1, k_2) \tag{4}$$

where M is the co-use matrix of peer P and N_{k_1,k_2} is the number of key couples. To have an overall perspective about the network state, the index is averaged over the N_p peers: $O_m = 1/N_p \cdot \sum_P O_m(P)$.

The *similarity* of a peer P is defined as the average value of the co-use frequency evaluated for key couples in which one key is stored in the local peer and the other is stored in $\mathcal{A}(\mathcal{P})$, the set of peers adjacent to P:

$$S_n(P) = \frac{1}{N_{k_a,k_b}} \cdot \sum_{k_a in P, k_b in \mathcal{A}(\mathcal{P})} M(k_a, k_b) \tag{5}$$

Again, the value is averaged over the whole network, $S_n = 1/N_p \cdot \sum_P S_n(P)$.

In Figs. 5 and 6, these indices are plotted versus time, starting from the instant in which the process is initiated and ant agents begin to travel the network. The two figures correspond to the two sets of experiments: the ProM logs and the traces generated with the myExperiment workflow pattern. In both cases, homogeneity and similarity increase rapidly and then get stabilized at values that are much higher than at the beginning. This trend confirms that keys are clustered and sorted by agents, which is the condition that allows query agents to find target keys, as described in Section 3.3.

It should be remarked that the transient phase occurs only once, when the process is initiated starting from a completely disordered system. After this phase, the indices are stable because every change in the network – connections and disconnections of peers, publishing and removals of resources – is rapidly tackled by ant agents. For example, when a peer publishes some new keys, the ant agents quickly notice that those keys are not in order, and use pick and drop operations to restore the order.

[3] http://prom.sourceforge.net/.

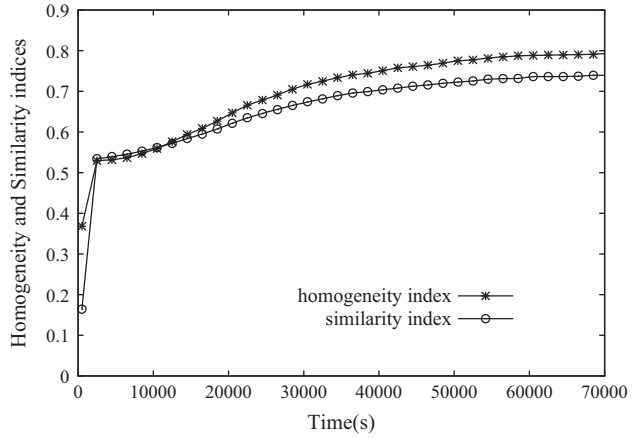

Fig. 5. Homogeneity vs. time, starting from a disordered network. Experiment executed with ProM logs.

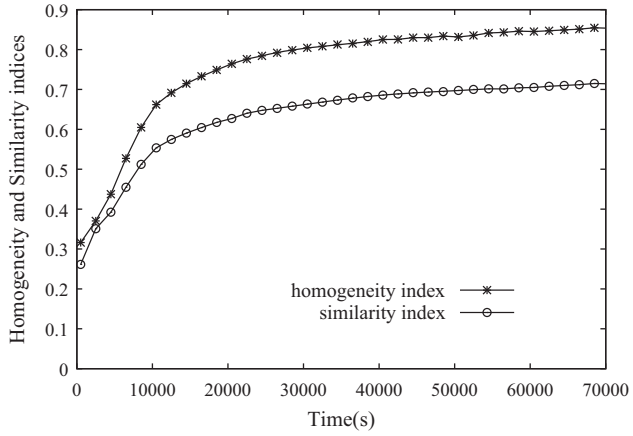

Fig. 6. Similarity vs. time, starting from a disordered network. Experiment executed with the myExperiment workflow pattern.

4.2. Performance of resource discovery

The reorganization of keys ensures that the descriptors of services included in a collective discovery request are located in the same or in neighbor peers: this fact is exploited by the discovery process. To evaluate the effectiveness of resource discovery, we computed the *percentage of "satisfied" service classes*, i.e., the percentage of service classes, specified in a workflow request, for which a minimum number of keys are discovered. This minimum number is a threshold used as a parameter, and is referred to as T_q.

Figures 7 and 8 report the values of this index vs. time when T_q is set to 2, 4 and 8. Again, the two figures correspond to the two sets of experiments. The dashed and continuous curves report, respectively, the percentage of service classes that are satisfied by the first query agent and at the end of the process. Here it may be useful to recall (see Section 3.3) that new query agents are issued by the requesting peer until the entire discovery request has been satisfied (T_q keys have been found for each service class) or it proves impossible to fulfill the goal (the last query agent does not discover any further key). The figures show that the index increases with time and, once the keys have been reorganized, the first query agent is sufficient to satisfy a large percentage of service classes, always higher than 70%. The percentage depends on the value of T_q: it is lower with higher values of T_q, because more keys are needed to satisfy the request. The two figures also show that at the end of the process the percentage of satisfied

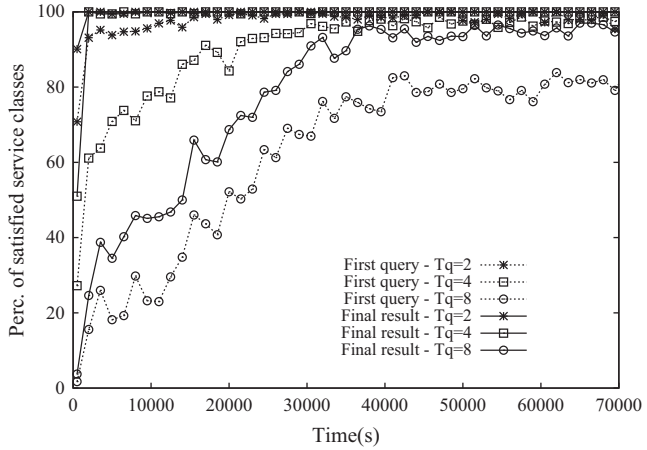

Fig. 7. Percentage of service classes satisfied by the first query agent (dashed lines) and at the end of the discovery process (continuous lines), for different values of the threshold T_q. Experiment executed with ProM logs.

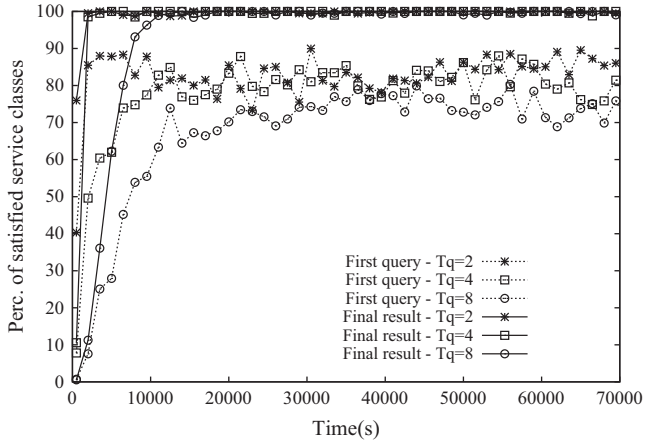

Fig. 8. Percentage of service classes satisfied by the first query agent (dashed lines) and at the end of the discovery process (continuous lines), for different values of the threshold T_q. Experiment executed with the myExperiment workflow pattern.

service classes is practically equal to 100%, meaning that all collective requests are successful.

The efficiency of the discovery process can be evaluated by assessing the number of issued query agents and the length of their path. Accordingly, we measured the average values of (i) the number of query agents needed by a discovery request to collect at least T_q keys for each service class; (ii) the overall number of steps performed by those query agents to fulfill the goal. Both indices should be minimized to reduce the response time experienced by the user and limit the traffic load. Figures 9–12 show that both indices decrease as the agents operate, proving that the reorganization of keys not only improves the effectiveness of the discovery process, but also its efficiency. In steady conditions, the average number of needed query agents (Figs. 9 and 10) is slightly higher than one, meaning that in most cases one or two query agents are sufficient to satisfy a collective discovery request. At the same time the overall number of steps performed by query agents (Figs. 11 and 12) is reduced to about 13 hops in the ProM case and 8 hops for the myExperiment scenario.

It is useful to compare the efficiency of this approach with that of typical structured P2P systems. In these systems, the keys are sorted over the network and can be typically reached in a number of steps that is logarithmic with the size of the network. However, as discussed before, the keys are dispersed by the hash function, therefore one separate query must be issued for each key included

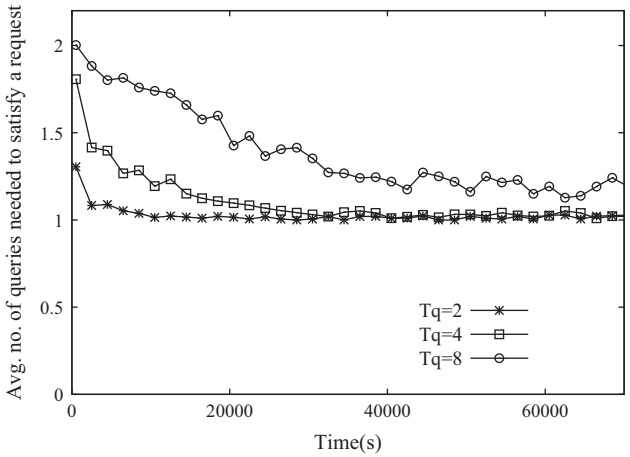

Fig. 9. Average number of query agents needed to complete a collective request, for different values of T_q. Experiment executed with ProM logs.

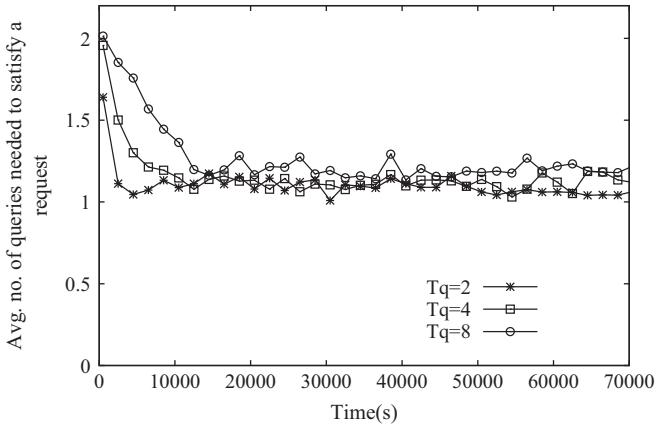

Fig. 10. Average number of query agents needed to complete a collective request, for different values of T_q. Experiment executed with the myExperiment workflow pattern.

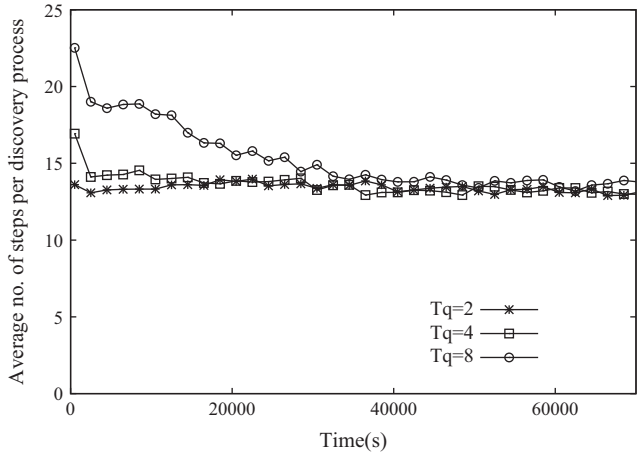

Fig. 11. Average number of steps performed by query agents to serve a collective request, for different values of T_q. Experiment executed with ProM logs.

in the collective request. In the two analyzed scenarios, myExperiment and ProM, the average number of target keys included in a discovery request is, respectively, 6.7 and 6. The first value was computed directly from the ProM logs, while the second value derives from the observation that the workflow pattern shown in Fig. 2 always needs the execution of 6 services. The number of hops to reach a key in a structured P2P systems with 2500 peers is

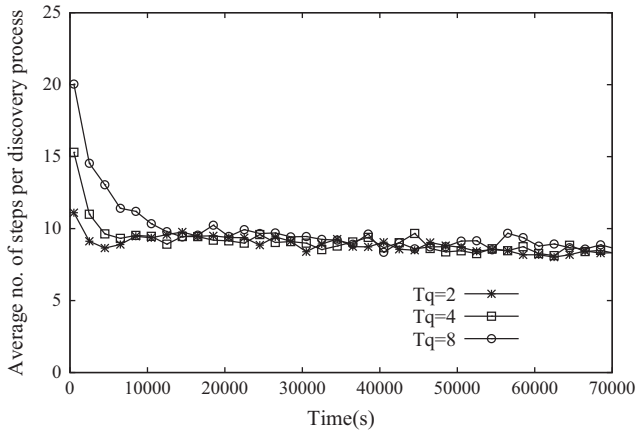

Fig. 12. Average number of steps performed by query agents to serve a collective request, for different values of T_q. Experiment executed with the myExperiment workflow pattern.

$\log_2(2500)$, i.e., about 11.3. It is then found that the number of hops performed by query agents is about $6.7*11.3 = 75.7$ for the myExperiment case (against 13 with our approach) and $6*11 = 66$ for the ProM case (against 8).

4.3. Analysis of computing and traffic load

This section discusses the computing and traffic load induced by ant agents and query agents. Ant agents are discussed first. The arrival of an ant agent at a peer triggers the execution of pick/drop operations and the update of the co-use matrix with the Push-Sum protocol. Therefore it is important to estimate the average number of agents per second that arrive and are processed at a peer. The arrival rate of agents to peers is given by the product of the average number of circulating agents $\overline{N}_a$ by the frequency of their movements $1/T_{mov}$, where T_{mov} is the mean of the random interval from the instant in which a peer receives an ant agent to the instant in which it forward the agent to the next peer. In this work, T_{mov} is set to 3 s. As discussed in Section 3.2, $\overline{N}_a$ and $\overline{N}_p$ ($\overline{N}_p$ is the average number of connected peers) are comparable. Thus, the arrival rate at a single peer can be computed as

$$\frac{\overline{N}_a}{\overline{N}_p \cdot T_{mov}} \approx \frac{1}{T_{mov}} \tag{6}$$

Therefore the arrival frequency of ants is comparable to the inverse of T_{mov}, i.e., one ant every 3 s. This result was confirmed by simulation experiments. The operations performed when receiving an ant are simple and fast, therefore the induced computing load is acceptable. From expression (6) it is also evident that the arrival frequency does not depend on the network size, which is a sign of good scalable behavior.

The traffic load induced by ant agents can also be estimated. Each ant carries the co-used matrix taken from the last peer. Since the co-use matrix is sparse and symmetrical, only a subset of its elements needs to be carried. In the ProM tests, the size of the matrix is 12×12, but on average only 40 non-zero elements need to be carried. In myExperiment tests, the size of the matrix is 20×20, and on average only 64 elements are significant. In both cases, the size of data carried with the co-use matrix is negligible. An ant also carries a number of keys taken with pick operations, along with the corresponding resource descriptors, each of which needs about 4 KBytes of data. The average number of carried keys is about 4 for both experiments, which means that an ant carries on average 16 KBytes of data. If this quantity is multiplied by the arrival frequency estimated before, it is obtained that the rate of

incoming data at a single peer is about $16/T_{mov}$ KBytes/s, i.e., about 5 KBytes/s.

The load induced by query agents is now examined. At the arrival of a query agent, some elements of the co-use matrix are updated (see Section 3.1) and the discovery algorithm is executed (Section 3.3). The arrival frequency of query agents can be obtained multiplying the frequency at which collective requests are issued by the average number of steps performed by query agents to serve a request. The first quantity depends on the behavior and activity of users. In the experiments, it is assumed that the users working on a single peer need to execute a workflow, and issue a corresponding collective request, every 1000 s, i.e., every 16.6 min. The average number of steps was reported in Figs. 11 and 12. Thus, the number of query agents that arrive per second at a single peer, for ProM and myExperiment tests is, respectively, about 13/1000 and about 8/1000: in both cases, less than one query agent per minute. The computing load caused by query agents is therefore acceptable. To obtain the traffic load induced by query agents, the arrival frequency must be multiplied by the size of data carried by a single query agent. The average number of keys carried by a query agent is about 18 for the ProM case and 30 for the myExperiment case. The size of the metadata descriptor associated with each key is about 4 KBytes, therefore the rate of incoming data is about (13/1000)∗18∗4 KBytes/s=0.93 KBytes/s with ProM, and about (8/1000)∗30∗4 KBytes/s=0.96 KBytes/s with myExperiment.

If a structured P2P system were adopted instead of the approach presented here, the computing and traffic load could be estimated in a similar way. Recalling the considerations made in Section 4.2, the arrival frequency of query agents would be about 75.7/1000 for the ProM case and about 66/1000 for the myExperiment case. Of course, a correspond increase in the traffic load would be observed. It can be concluded that the computing and traffic load induced by the presented algorithms is lower than the load experienced in classical P2P systems.

5. Conclusion

This paper presented a self-organizing peer-to-peer framework and a set of nature-inspired algorithms that aim to assist the user in the design and execution of complex applications and workflows of services. The main idea is to place the descriptors of services in the same or in close-by peers if they are often used together, so as to facilitate the collective discovery of the basic services needed to compose a workflow. The relocation of services is performed with the help of mobile agents, which move the service descriptors over the network, and is assisted by statistical information about the co-occurrence of services in past workflow instances. Performance evaluation focused on two scenarios taken from the literature, specifically from ProM and myExperiment scientific projects. Results show that the services are clustered and sorted by mobile agents. This allows the efficacy and efficiency of discovery operations to be notably improved, while the computing and network load induced by the algorithms is moderate and easily sustainable.

References

Barabási A-L, Albert R. Emergence of scaling in random networks. Science 1999;286 (5439):509–12.

Birukou A, Blanzieri E, Giorgini P, Kokash N. Improving web service discovery with usage data. IEEE Software 2007;24:47–54.

Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence: from natural to artificial systems. New York, NY, USA: Oxford University Press; 1999.

Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 2009;25(6):599–616.

Cheema AS, Muhammad M, Gupta I. Peer-to-peer discovery of computational resources for grid applications. In: Proceedings of the 6th IEEE/ACM international workshop on grid computing. Seattle, WA, USA; 2005. p. 179–85.

Di Bernardo M, Pottinger R, Wilkinson M. Semi-automatic web service composition for the life sciences using the biomoby semantic web framework. Journal of Biomedical Informatics 2008;41(5):837–47.

Doerner K, Merkle D, Sttzle T. Special issue on ant colony optimization. Swarm Intelligence 2009;3(1):1–2.

Dustdar S, Papazoglou MP. Services and service composition – an introduction. Information Technology 2008;50(2):86–92.

European Community for Software & Software Services, January 2009. White paper on software and service architectures, infrastructures and engineering – action paper on the area for the future EU competitiveness. Technical Report. Available at: ⟨http://www.eu-ecss.eu/documentation/ecss-documentation⟩.

Forestiero A, Mastroianni C, Spezzano G. Reorganization and discovery of grid information with epidemic tuning. Future Generation Computer Systems 2008a;24(8):788–97.

Forestiero A, Mastroianni C, Spezzano G. So-grid: a self-organizing grid featuring bio-inspired algorithms. ACM Transactions on Autonomous and Adaptive Systems 2008b;3(2):5:1–37.

Forestiero A, Leonardi E, Mastroianni C, Meo M. Self-chord: a bio-inspired P2P framework for self-organizing distributed systems. IEEE/ACM Transactions on Networking 2010a;18(5):1651–64.

Forestiero A, Mastroianni C, Papuzzo G, Spezzano G. A proximity-based self-organizing framework for service composition and discovery. In: Proceedings of the 10th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid 2010). Melbourne, Australia; 2010b. p. 428–37.

Handl J, Meyer B. Ant-based and swarm-based clustering. Swarm Intelligence 2007;1(2):95–113.

Kempe D, Dobra A, Gehrke J. Gossip-based computation of aggregate information. In: Proceedings of the 44th annual IEEE symposium on foundations of computer science; 2003.

Maamar Z, Faci N, Wives LK, Badr Y, Santos PB, deOliveira JPM. Using social networks for web services discovery. IEEE Internet Computing 2011;15:48–54.

Meng H, Wu L, Zhang T, Chen G, Li D. Mining frequent composite service patterns. In: GCC '08: proceedings of the 2008 seventh international conference on grid and cooperative computing; 2008.

Meyer L, Annis J, Wilde M, Mattoso M, Foster I. Planning spatial workflows to optimize grid performance. In: SAC'06: proceedings of the 2006 ACM symposium on applied computing, New York, NY, USA: ACM; 2006. p. 786–90.

Papazoglou MP, Traverso P, Dustdar S, Leymann F. Service-oriented computing: state of the art and research challenges. Computer 2007;40(11):38–45.

Pilioura T, Kapos G-D, Tsalgatidou A. PYRAMID-S: a scalable infrastructure for semantic web service publication and discovery. In: RIDE '04: Proceedings of the 14th international workshop on research issues on data engineering. Boston, MA, USA; 2004.

Rao J, Su X. A survey of automated web service composition methods. In: Workshop on semantic web services and web process composition, SWSWPC, LNCS, vol. 3387. Springer: San Diego, CA, USA; 2004. p. 43–54.

Rasch K, Li F, Sehic S, Ayani R, Dustdar S. Context-driven personalized service discovery in pervasive environments. World Wide Web 2011;14:295–319.

Rios J, Karlsson J, Trelles O. Magallanes: a web services discovery and automatic workflow composition tool. BMC Bioinformatics 2009;10(1):334.

Riteau P, Tsugawa M, Matsunaga A, Fortes J, Keahey K. Large-scale cloud computing research: sky computing on futuregrid and grid 5000. ERCIM News 2010;83 (83):41–2.

Sellami M, Tata S, Maamar Z, Defude B. A recommender system for web services discovery in a distributed registry environment. In: Proceedings of the 2009 fourth international conference on internet and web applications and services. Washington, DC, USA: IEEE Computer Society; 2009. p. 418–23.

Srivastava B, Koehler J. Web service composition – current solutions and open problems. In: Proceedings of the ICAPS 2003 workshop on planning for web services. Trento, Italy; 2003. p. 28–35.

Taylor IJ. From P2P to web services and grids: peers in a client/server world. Springer; 2004.

van der Aalst WMP, Reijers HA, Weijters AJMM. Business process mining: an industrial application. Information Systems 2007a;32(5):713–32.

van der Aalst WMP, van Dongen BF, Günther CW, Mans RS, Alves de Medeiros AK, Rozinat A, et al. Prom 4.0: comprehensive support for real process analysis. In: 28th international conference on application and theory of petri nets and other models of concurrency, ICATPN 07, Siedlce, Poland; 2007b. p. 484–94.

Verma K, Sivashanmugam K, Sheth A, Patil A, Oundhakar S, Miller J. METEOR-S WSDI: a scalable P2P infrastructure of registries for semantic publication and discovery of web services. Information Technology and Management 2005;1:17–39.

Wisner P. Automatic composition in service browsing environments. In: MIRW 2006, Workshop on mobile interaction with the real world. Espoo, Finland; 2006.

Zimeo E, Troisi A, Papadakis H, Fragopoulou P, Forestiero A, Mastroianni C. Cooperative self-composition and discovery of grid services in P2P networks. Parallel Processing Letters 2008;18(3):329–46.