Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 512 – 519

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

Using Service Clustering and Self-Adaptive MOPSO-CD for QoS-Aware Cloud Service Selection

Giandomenico Spezzano

[a]*CNR-ICAR, Via P. Bucci 7-11c, Rende (CS), Italy*

Abstract

A promising way to effectively manage the composition of services in a heterogeneous and dynamic environment is to make workflow management able to self-adapt at runtime to react to changes in its environment by autonomously reconfiguring itself. Most of the proposed methodologies address this issue as a QoS-aware *service selection* problem in which a web service broker can dynamically select the "right" service that takes part in the composition, and adaptively change the bound service when the delivered QoS has changed.

In this paper, we propose a self-organizing framework that uses a service clustering based discovery approach to effectively and efficiently support the selection of services in which runtime changes in the QoS of the services are taken into account. Two bio-inspired algorithms are designed to support a QoS-aware dynamic service selection mechanism. An ant-based clustering algorithm enhanced with a template mechanism that guides the artificial ants to move data items to construct and maintain a specific topology is adopted as a method for efficient service discovery. As a consequence, services can dynamically be discovered in a shorter time and with lower network traffic. To select the actual concrete services that best meet the user QoS requirements a Self-Adaptive Multi-Objective Particle Swarm Optimization Algorithm using crowding distance technique (MOPSO-CD) is executed using the topological map generated by the ant algorithm. In the end, simulation results show the effectiveness of the method proposed.

Keywords:
QoS-aware service selection; Ant-based service clustering; Particle swarm optimization; Cloud computing

1. Introduction

Cloud computing[1] has emerged as a global platform that provides a service-oriented infrastructure (SOA) that uses standardized protocols, allows users to have seamless access, and coordinated sharing of distributed computing resources providing various services. Implementation of SOA in a Cloud environment brings about challenges which

* Corresponding author. Tel.: +39-0984-493854; fax: +39-0984-493010
E-mail address: spezzano@icar.cnr.it

doi:10.1016/j.procs.2016.04.245

include service discovery, service selection, service composition, robustness, Quality of Services, etc. These challenges are mainly due to the dynamic nature of the operational environment. SOA may often need to dynamically (re)-organize its topologies of interactions between the services due to unpredictable events, such as crashes or network problems, which will cause services unavailability. The dynamic characteristic of SOA is quite similar with the characteristic of self-organizing systems, in the way that they are able to organize elements (services in the case of SOA) in order to change their functions or create new functions on higher levels (emergence). Therefore, we believe that the dynamic characteristic of SOA can benefit from the use of self-organization primitives found in nature. In this paper, we elaborate the idea of adapting self-organizing systems found in nature into SOA in order to ensure the robustness of SOA and solve the service selection problem[2] in cloud service composition.

In a Cloud service composition system[3] web services can be selected and integrated to generate a workflow for the satisfaction of user functional requirements. Usually, for each task in the workflow a set of alternative web services with similar functionality is available, and these web services have different QoS parameters. A mechanism to dynamically select the best or near-best services according to the QoS measured have to be adopted to enable the workflow to be adaptive. This leads to the general optimization problem of how to select web services for each task so that the overall QoS requirements of the composition are satisfied and optimal.

Our focus is on selecting, from functionally capable services, the best services in terms of their non-functional attributes according to the changing service environment. Our methodology is based on the discovery of relevant web services through the creation of cluster services that group web services that provide similar functionality on which to apply a QoS-aware service selection heuristic local optimization method.

We propose a self-organizing framework to support service selection for QoS-aware dynamic service composition that uses two bio-inspired algorithms to enable the system to be adaptive, and thus effective in highly dynamic scenarios. An ant-based algorithm is adopted as a method for service clustering and sorting. Its role is that to identify dynamic service pools of functionally equivalent candidate concrete services. The neighbourhood relation between clusters is based on a predefined pattern known as template. The template serves as a mechanism that guides the ants to build a specific topology between clusters. The second algorithm is a novel global QoS optimizing and multi-objective web services selection algorithm based on particle swarm optimization (PSO)[7]. Its role is to determinate, from the service pools identified by the ant algorithm, the concrete services which are bound to each incoming user request while meeting user QoS requirements.

The rest of the paper is organized as follows. Section 2 discusses briefly the self-organizing service selection approach adopted. Section 3 presents the ant-based algorithm enhanced with a template mechanism for service clustering according to a specific topology. Section 4 presents briefly the particle swarm optimization (PSO). Section 5 describes the MOPSO-CD algorithm that use a multi-objective strategy to support the selection of service. The results are presented and discussed in Section 6.

2. Overview of Our Approach

With the increase in the number of services, the whole performance of Web service composition will be affected, thus the study of web service composition based on QoS service selection *intelligent* optimization algorithm and its realization is very important.

A straightforward method for finding the optimal composition is enumerating and comparing all possible combinations of candidate services. For a composition request with n service classes and l candidate services per class, there are l^n possible combinations to be examined. Hence, performing an exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for run-time service selection in applications with many services and dynamic needs. To effectively deal with the large number of services is often impossible to analyse whole services. Only the web services belonging to service class of interest would be selected for analysis while the web services belonging to the remaining classes could be discarded.

Furthermore, during the execution of a composite service, some component web service may update their QoS property on-the-fly, others may become unavailable, and still others may emerge. Consequently, approaches where web services are statically composed are inappropriate. Instead, a dynamic composition approach is needed, in which runtime changes in the QoS of the component services are taken into account.

Our approach combines monitoring (*dynamic service clustering*) into an automating service selection methodology that is robust in the face of varying QoS. For an efficient service selection method, it is necessary that the algorithm is able to automatically discover services and choose between a set of similar services. Putting these services into a cluster considering their functional features can reduce the problem-scale of QoS-based selection service.

The framework here proposed is intended to provide an effective means for dealing with QoS-based service selection in highly dynamic scenarios. The framework introduces two bio-inspired algorithms that work together to enable the system to be adaptive. Figure 1 shows how the framework helps client applications to select the invoked service dynamically according to the changing service environment.

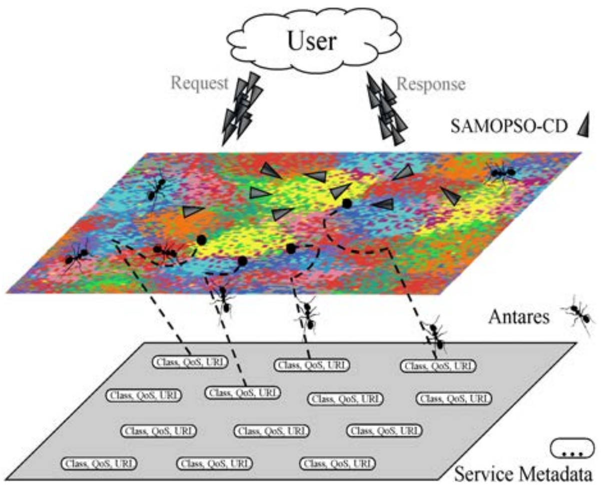


Fig. 1. QoS-based Web service selection Framework.

At low level, a P2P distributed repository provides a service metadata container capable of storing information about services. The metadata constitutes the basic information of concrete services that are managed by framework. A concrete service is represented by a descriptor that contains all information that is necessary to identifies the membership class of the service. Local service information can be injected (and updated upon significant changes in QoS value) in the repository to contribute to the dynamic formation of a distributed ”UDDI”, that can be represented as a sort of virtual landscape of the distribution of service descriptors on the network. In this way, the discovery of the services on the network is reduced to the problem of finding service descriptors, through which it is possible to access the corresponding services. To make efficient the discovery of the services it may/might be useful to group all the services according to their membership class. Data mining techniques, especially ant-based clustering analysis, can be used to group and sort similar services. As a consequence, services can dynamically be discovered in a shorter time and with lower network traffic. We use ANTARES system[5] to create and sort dynamic service clusters.

The selection phase is based on local selection of services using a novel Self-adaptive Multiple Objective Particle Swarm Optimization with Crowding Distance technique (MOPSO-CD) to find the best match between the client requests and the service providers. MOPSO-CD considers topographic map and dominance relationships between available equivalent services to select the best services to be considered as candidates for composite process.

Next sections describe the ANTARES and MOPSO-CD algorithms used to implement the QoS-aware selection service framework.

3. The ant-based service clustering algorithm

ANTARES is a novel approach for the construction of a Cloud information system, which is inspired by the behaviour of some species of ants. ANTARES is able to disseminate and reorganize descriptors and, as a consequence of this, it facilitates and speeds up discovery operations. More specifically, ANTARES concurrently achieves the following objectives:

- it replicates and disseminates descriptors, with the possibility of fostering the dissemination of descriptors associated to dynamic and/or high QoS resources;
- it *spatially* sorts descriptors, so that descriptors indexed by similar keys are placed in neighbour hosts;
- thanks to the self-organizing nature of the ant-based approach, the reorganization of descriptors spontaneously adapts to the ever changing environment, for example to the joins and departs of hosts and to the changing characteristics of resources.

A prerequisite for applying ANTARES to web services is to establish a method for evaluating the similarity level between two services. We consider that two services are similar if there a high degree of match between their descriptions. In order to facilitate the selection of services we adopt a web service description model where each web service can be described as WS = {*Class*, *QoS*, *URL* }, where *Class* is function attribute set of service, *QoS* is quality of service attribute set, *URL* is the address need to reference the Web service. According to *Class* service requestors determine whether the services meet the needs of its function, and according to *QoS* select suitable services from a number of candidate services in one service. *Class* uses a syntactic approach to map a WSDL description in an index (descriptor) where each characteristic is represented as a dimension of an n-dimensional vector space. ANTARES can easily create clusters of Web services since has been specifically designed to tackle the case in which the access keys of resource descriptors are bit strings and, since similar resources are assumed to be mapped into similar strings, it is possible to define a similarity measure among resources, through the comparison of related keys.

The ant-inspired agents of ANTARES replicate and move descriptors, and tend to place descriptors with similar (or equal) keys into the same host or neighbour hosts. This is achieved by ants pick and drop operations which are driven by corresponding pick and drop probability functions. The obtained rearrangement and spatial ordering of descriptors facilitates resource discovery and enables range queries.

The behaviour of ants is imitated by mobile agents that, while hopping from one peer to another, can copy and move service descriptors. Given a set of n-dimensional data represented as points in a n-dimensional space, and a metric *d* which measures the distance between pairs of data items, project the points on a plane so that points in the plane belong to a same cluster if and only if the corresponding data items are similar in the n-dimensional space under the metric *d*. The ants move on the plane and possibly pick an object or drop an object. The movement of the ant is random. On reaching the new location on the plane the ant may possibly pick up an object or drop an object, if it is carrying one. The heuristics and the exact mechanism for picking up or dropping an object are explained below.

ANTARES progressively and continuously constructs service clusters by a number of ant-inspired agents which travel the hosts through P2P interconnections, possibly pick service descriptors from a host, carry these descriptors, and drop them into other hosts. The algorithm has been specifically designed to *sort* service descriptor in order to facilitate and speed their discovery.

3.1. Picking up an object

When the ant is not carrying any object, it looks for possible objects to pick up by looking at the neighbourhood around its current position. Actually, the agent evaluates the similarity of the binary key of the descriptor under consideration with the keys of the centroids of the current host and of the neighbour hosts, and then takes the average. A centroid of a host is a virtual descriptor whose key is representative for the descriptors maintained in the local host. Similarity is evaluated against centroids (instead of against all single descriptors) in order to reduce the information exchanged among hosts. The pick probability function P_{pick}, is defined in formula (1) whereas f, defined in formula (2), measures the average similarity of a generic descriptor $\overline{d}$ with the other descriptors located in the visibility region R. In more detail, in formula (1) the parameter k_p, whose value is comprised between 0 and 1, can be tuned to modulate the degree of similarity.

$$P_p = (\frac{k_p}{k_p+f})^2 \qquad (1)$$

In formula (2), the weight of each term is equal to the number of descriptors N_p maintained in each peer p, while N is the overall number of descriptors maintained in the region R.

$$f(d,R) = \frac{1}{N} * \sum_{p \in R} N_p * (1 - \frac{1-cos(d,C_p)}{\alpha}) \qquad (2)$$

Note that the similarity between d and the descriptor of the centroid of the peer p, C_p, is defined as the cosine of the angle between the corresponding key vectors. The parameter α defines the similarity scale; here it is set to 0.5. Actually, with the adopted values of α and k_p, the value of f can range between -1 and 1, but negative values are truncated to 0: this corresponds to have, in formula (1), a P_{pick} value of 1 in the case that the evaluated descriptor is very dissimilar from the other descriptors.

The pick operation can be performed with two different modes, *copy* and *move*. If the *copy* mode is used, the agent, when executing a pick operation, leaves the descriptor on the current host, generates a replica of it, and carries the new descriptor until it drops it into another host. Conversely, with the *move* mode, an agent picks the descriptor and removes it from the current host, thus preventing an excessive proliferation of replicas.

3.2. Dropping an object

When the ant is carrying an object, then it examines the neighbours of its current location. For each carried descriptor, the agent separately evaluates the drop probability function, which, as opposed to the pick probability, is directly proportional to the similarity function f defined in formula (2), i.e., to the average similarity of this descriptor with the descriptors maintained in the current visibility region. In (3), the parameter k_d is set to a higher value than k_p, specifically to 0.5, in order to limit the frequency of drop operations. As for the pick operation, the agent first evaluates P_{drop}, then extracts a random real number between 0 and 1, and if the latter number is lower than P_{drop}, the agent drops the descriptor in question into the current host.

$$P_d = (\frac{f}{k_d+f})^2 \quad (3)$$

Note that an agent must drop all the descriptors that it maintains before it can try to pick other descriptors from a new host. The inverse and direct proportionality with respect to the similarity function f assures that, as soon as the possible initial equilibrium is broken (i.e. descriptors having different keys begin to be accumulated in different regions, the reorganization of descriptor is more and more facilitated.

3.3. Spatial constraints using template

The ant clustering shows some spatial self-organizations but has the specificity to generate clusters at random places. According to the first random moves that the ants start to do in the beginning of the algorithm, some material will initiate aggregation and the clustering process will complete this aggregation from these initial random first aggregations. It has been recognized that in ant species such as Leptothorax albipennis[4], ants construct the nest wall based on a predefined pattern known as template. The template serves as a blueprint that directs the ants to build the nest in a predefined size and shape, hence constraining the self-organizing dynamics of aggregate behaviors. Franks and Deneubourg in[4] proposed a mechanism that combines self-organization and templates (Ant_TM) to specify the picking-up and dropping behaviours.

Distance Hamming-based templates are successfully used to supervise artificial ants on dropping data items in a specific region of a 2-dimensional space. The template probability P_t(i,j) is defined as:

$$P_t(i, j) = a * [\sum_{k=0}^{n-1} (y_{i,k} \neq y_{j,k}) - b]$$

In the equation $P_t(i, j)$ is the Hamming distance between the objects i and j, k is the index of the respective variable reading y out of the total number of variables n. The Hamming distance itself gives the number of mismatches between the variables paired by k. The a and b parameters are used to make $0 \leq P_t(i, j) \leq 1$. Using this template function, we replace in the clustering algorithm, the two previous probabilities defined in equation (1) and equation (3) by:

$$P_p^{'} = c_1 P_p + c_2(1 - P_t)$$
$$P_d^{'} = P_d * P_t$$

where c_1 and c_2 are two constants to set the relative importance of ant-based clustering and template mechanism for picking up, where $c_1 + c_2$ is always equal to 1, and i and j determine the coordinates of the ant on the grid.

The picking up possibility $P^{'}_{p}$ is defined as linear combinations, and the dropping possibility $P^{'}_{d}$ is defined as a product. By defining this way, artificial ants can favour those data items that stay at frontier areas of dropping regions when picking up data items. And while dropping data items, artificial ants prefer to drop data items at the area close to the value of the template deployed on the grid. This strategy results in the effect that similar data items tightly stay together around the template.

4. Introduction to PSO

Particle swarm optimization (PSO) algorithm has been developed by Eberhart and Kennedy in 1995[6] and it is a population based stochastic optimization technique inspired by social behaviour of bird flocking or fish schooling. The system is initialized with a population of random solution and searches for optimum by updating generations. The potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. The value of fitness is also stored. This value is called *Pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbours of the particle. When a particle takes all the population as its topological neighbours, the best value is a global best and is called *Gbest*.

The particle swarm optimization concept consists of, at each time step, changing the velocity of each particle towards its pbest and gbest locations according the following formulas:

$$v_{k+1} = c_0\, v_k + c_1(pbest_k - x_k) + c_2(gbest_k - x_k)$$
$$x_{k+1} = x_k + v_{k+1}$$

where v_k is the velocity vector of a particle; x_k is the current location; c_0 is the inertia factor, parameters c_1 and c_2 are random numbers and represent the cognitive and social learning rates, respectively.

5. The Self-Adaptive MOPSO-CD algorithm

The problem of cloud service selection consists of finding an efficient algorithm which can match client requests and services efficiently, while optimizing the multiple objectives defined by QoS parameters at the same time. PSO has proved to be an efficient optimization method for single objective optimization. However the basic form of PSO has some serious limitations in particular when dealing with multi-objective optimization problems.

The proposed algorithm which we shall call Self-Adaptive MOPSO-CD extends the algorithm of a single-objective PSO to handle multi-objective optimization problems. A description of the MOPSO-CD algorithm used to optimize the cloud service selection method is given in listing 1.

MOPSO-CD starts with a population *P* of completely randomly generated particles. In each generation, the fitness of each population member is evaluated. For every generation, each particle moves on the virtual clustered landscape created by ANTARES toward the region (*cluster*) containing the descriptors belonging to the class of service requested by computing the Hamming distance between the class of the requester-service descriptor achieved so far and that of the new position. If the Hamming distance of the particle of the new position is less than or equal to the current then the *pbest* of the particle is updated.

All the classes of the new solutions found in *P* are stored into an external archive (*EleSol*). When the Hamming distance is equal to zero and the new position is an eligible solution (i.e. the QoS vector of the new solution dominates the requested QoS vector), then the eligible solutions found in P are stored in an another external archive *NoDom*. All new eligible solutions found are inserted in an external archive that stores non dominated solutions (*NoDom*) if they are non dominated by any of the stored solutions.

The main objective of the external archive *NoDom* is to record non-dominated solutions found along the search process. The algorithm has to decide whether a certain solution should be added to the archive or not. Initially, non-dominated solutions are added to the *NoDom* archive. After each iteration, non dominated solutions of the swarm are compared with the solutions in the external archive. The candidate solutions that are not dominated by the solutions of the external archive are added to it. Then, the dominated solutions are removed from the external archive. The external archive has a maximum number of solutions. If this number is exceeded after the end of each iteration, solution in more crowded regions are removed from the repository using the crowding distance criterion.

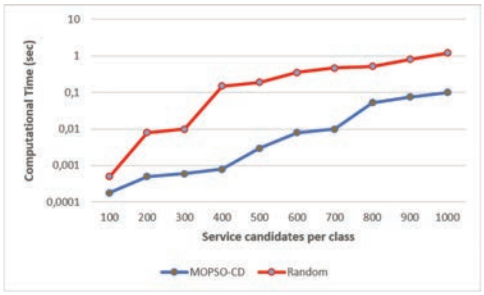
Listing 1. The MOPSO-CD algorithm

```
Step 1. Initialize population

for i = 0 to M
    // (M is the population size)
    Initialize P[i] randomly
    // (P is the population of particles)
    Initialize V[i] = 0
    // (V is the speed of each particle)
    Evaluate P[i]
    Initialize the personal best of each particle
    PBESTS[i] = P[i]
    GBEST = Best particle found in P[i]
end for

Step 2. Iterate to find the solutions

    Initialize the iteration counter t=0
    repeat
      for i = 1 to M
       if (HammingDistance(P[i].class, QueryRequested.class) >= HammingDistance (P[i].pbest.class, QueryRequested.class))
              PBEST[i]= P[i]
              Store the class of the solution found in P into EleSol List
              // (EleSol List is the external archive that stores solutions found in P)
       if (HammingDistance(P[i].class,QueryRequested.class) = 0 & IsElegibleSolution(P[i].QoS, QueryRequested.QoS)
              Store the elegible solutions found in P into NoDom List
              // (NonDom List is the external archive that stores nondominated solutions found in P)
      end for
        Insert all new elegible solutions in P in NoDom if they are not dominated by any of the stored solutions.
        Compute the minimum value of the class values of the solutions in the archive EleSol
        Remove all solutions whose class values are greater than the minimum in EleSol
        Compute the crowding distance values of each solution using the x and y coordinates in the archive EleSol
        Sort the solutions in \emph{EleSol} in descending crowding distance values
        Randomly select the global best for P[i] from a specified top portion (e.g. top 20 ) of the
        sorted archive Elesol and store its position to GBest
        Compute the standard deviation (stdev) of crowding distances
         if averageStdev < stdev
             inertia = function1()
           else
             inertia = function2()
         Update averageStdev with Stdev
         for i = 1 to M
           Compute the new velocity:
           Compute the new position
         end for
    until maximum number of iterations is reached
```

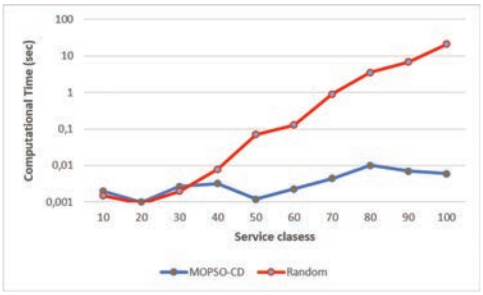
The minimum value of the class values stored in the *EleSol* archive is computed and all solutions whose class value are greater than the minimum in *EleSol* are deleted. The *Elesol* archive is updated considering a geographically-based system defined in terms of the objective function values for each particles. In this approach the space is explored using the crowding distance. The crowding distance is used as a mechanism to select the leaders from the external archive. The crowding distance value of a solution provides an estimate of the density of solutions surrounding that solution. The crowding distance values of each solution using the x and y coordinates in the *EleSol* archive is computed. Then the solutions contained in *EleSol* are sorted in descending distance values. The global best for P[i] is selected from a specified top portion (e.g. top 20) of the sorted archive EleSol and its position is stored to Gbest.

6. Simulation Results

To evaluate our proposed solution we implement a simulation using Netlogo[8] and we conducted several experiments, which we describe in this section. For the QoS data we use the QWS real dataset from[9]. This data set includes measurements of 9 QoS attributes for 364 real web services. To evaluate the scalability of our solution, however, we need to run experiments with a much larger set of services. Therefore, we duplicated the QWS dataset several times, each time multiplying the quality values of web services by a uniformly distributed random value between 0.1 and 2.0. In this way we achieved a data set of about 100.000 services. Figure 2 shows a comparison of the performance between the "random choice" approach (labeled with "Random") where the service selection is made at random and our approach (labeled with "MOPSO-CD"). We evaluate the performance of our approach, by measuring the required time for finding the solution (i.e. the best combination of concrete services). In this experiment we study the performance of our approach with respect to the size of the problem *m* in terms of the number of service classes *n* and the number of service candidates per class *l*. In the graph shown to the left of figure 2 we fix the number of service class *n* to 20 and vary the number of service candidates *l* from 100 to 1000 per class. In the right part of the graph the number of service candidates *l* is fixed to 100 and the number of service classes *n* varied from 10 to 100. The number of global constraints *m* in both cases is fixed to 3. The results in both graphs show that our approach has a good scalability in all problem instances (always less than 100 msec).



(a) 20 service classes - 3 constraints



(b) 100 service candidate per class - 3 constraints

Fig. 2. Computational time with respect to the problem size (Logarithmic scale is used).

7. Conclusions

In this paper, we have proposed a novel framework to support the selection of services in which runtime changes in the QoS of the services are taken into account. A self-organized approach is used to construct and maintain a specific topology for efficient service discovery. Such topology is used from the MOPSO-CD algorithm to navigate through the concrete services to find those with the best QoS. Experimental results show that the our approach has a good scalability.

References

1. Armbrust, M., Fox, A., Griffith, R., et al.: A view of cloud computing. *Communications of the ACM*, 53(4), pp. 5058, 2010.
2. Yau, S.S., Yin, Y.: Qos-based service ranking and selection for service-based systems. In: *IEEE International Conference on Services Computing (SCC)*, pp. 5663, 2011.
3. Wang, S., Zheng, Z., Sun, Q., Zou, H., Yang, F.: Cloud model for service selection. In: *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 666671. IEEE Press, 2011.
4. N. Franks and J. Deneubourg: Self-organizing Nest Construction in Ants: Individual Worker Behaviour and The Nests Dynamics. *Animal Behaviour*, 54:779796, 1997.
5. A. Forestiero, C. Mastroianni, G. Spezzano: Antares: an Ant-Inspired P2P Information System for a Self-Structured Grid. *BIONETICS 2007*: 151-158, 2007.
6. Kennedy, J., Eberhart, R.C.,: Particle Swarm Optimization In *Proceedings of IEEE International Conference on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, 5(3), pp 1942-1948, 1995
7. Nagy, A.; Oprisa, C.; Salomie, I.; Pop, C.B.; Chifu, V.R.; Dinsoreanu, M., "Particle Swarm Optimization for Clustering Semantic Web Services," in 10th International Symposium on Parallel and Distributed Computing (ISPDC), pp.170-177, 2011.
8. Wilensky, U., NetLogo. *http://ccl.northwestern.edu/netlogo/*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.
9. Al-Masri, E., and Mahmoud, Q. H., "QoS-based Discovery and Ranking of Web Services", IEEE 16th International Conference on Computer Communications and Networks (ICCCN), pp. 529-534, 2007.