Self-Chord: a Bio-Inspired P2P Framework for Self-Organizing Distributed Systems

Self-Chord: A Bio-Inspired P2P Framework for Self-Organizing Distributed Systems

Agostino Forestiero, *Associate Member, IEEE*, Emilio Leonardi, *Senior Member, IEEE*, Carlo Mastroianni, *Member, IEEE*, and Michela Meo, *Member, IEEE*

***Abstract*—This paper presents "Self-Chord," a peer-to-peer (P2P) system that inherits the ability of Chord-like structured systems for the construction and maintenance of an overlay of peers, but features enhanced functionalities deriving from ant-inspired algorithms, such as autonomous behavior, self-organization, and capacity to adapt to a changing environment. As opposed to the structured P2P systems deployed so far, resource indexing and placement is uncorrelated with network structure and topology, and resource keys are organized and managed by self-organizing mobile agents through simple local operations driven by probabilistic choices. Self-Chord has three main features that are particularly advantageous in Grid and Cloud Computing: 1) it is possible to give a semantic meaning to keys, which enables the execution of range queries; 2) the keys are fairly distributed over the peers, thus improving the balancing of storage responsibilities; 3) maintenance load is also limited because it is not necessary to reassign keys when new peers or resources are added to the system—the mobile agents will spontaneously reorganize the keys. The efficiency and effectiveness of Self-Chord were assessed both with a simulation framework and with an analytical model inspired by fluid dynamics.**

***Index Terms*—Bio-inspired algorithms, cloud computing, grid computing, multiagent systems, peer-to-peer (P2P), self-organization.**

I. Introduction

THE information service is an important component of distributed computing systems, such as computational Grids and Clouds, since it provides information about the resources that can be used to build and run complex applications and enables their discovery. Grids [1] use the resources of many networked computers to solve large-scale computation problems in multiple and heterogeneous domains. The large-scale and dynamic nature of Grids make human administrative intervention difficult or even unfeasible, and centralized information services are proving unsuitable to scale to hundreds or thousands of nodes. To tackle these issues, the scientific community has proposed to design information services according to the peer-to-peer (P2P) paradigm, which offers better scalability and adaptivity features [2]. A similar trend can be envisioned for the recently emerged Cloud paradigm [3], which is switching computation and storage responsibilities from the client size to the "clouds," i.e., to unseen computers on the server side, possibly scattered across continents. This paradigm is already adopted by Amazon for the on-demand provisioning of computing capacity and storage facilities, through Web services like the EC2 (Elastic Cloud Computing) and the S3 (Simple Storage Service). Grid and Cloud issues are similar in many aspects [4], especially in the need to assure scalability in a dynamic environment. Therefore, P2P techniques are very likely to be adopted in Clouds as they are today in Grids.

P2P models are classified into *unstructured* and *structured* based on the way nodes are linked to each other and data about resources is placed on the nodes [5]. In unstructured systems, resources are published by peers without any global planning. This facilitates network management, but reduces the efficiency of discovery procedures. In structured systems, resources are associated to specific hosts, often through *Distributed Hash Tables*. For example, in Chord [6], each peer is assigned a binary code, or "key," by a hash function, and peers are organized in a ring and ordered following the values of their keys. Resources are also indexed by keys, and each resource is consigned to the peer that has the same key as the resource or, if such a peer is not present in the ring, on the first following peer, also called "successor." Other structured P2P systems use different structures to organize the peers, but the basic principle is the same: Every resource is assigned to a well-specified peer on the structure. Structured systems are generally more efficient in terms of search time and network load, but, with respect to unstructured systems, can limit the expressiveness of discovery requests—users are only allowed to search for specific resources, but cannot issue complex or "range" queries. Moreover, structured systems may be difficult to administer in the case of high churn rate because new or modified resources must be immediately (re)assigned to the corresponding peers.

Along with the P2P approach, another interesting and recent trend is the design of *self-organizing Grids* [7], often inspired by biological systems such as ant colonies and insect swarms. Complex functionalities are achieved by mobile agents that perform simple operations at the local level, but at the global level engender an advanced form of intelligence that would be impossible to obtain with centralized or human-driven strategies. Bio-inspired techniques have already been exploited to solve a number of complex problems, such as task allocation, routing problems, graph partitioning, etc. [8]. Recently, these

Manuscript received March 06, 2009; revised October 02, 2009; accepted March 19, 2010. Date of publication April 19, 2010; date of current version October 15, 2010. This work was supported in part by the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). This work has been presented at the IEEE CCGrid 2009 Symposium.

A. Forestiero and C. Mastroianni are with the Institute for High Performance Computing and Networking, ICAR-CNR, 87036 Rende, Italy (e-mail: forestiero@icar.cnr.it; mastroianni@icar.cnr.it).

E. Leonardi and M. Meo are with the Department of Electronics of the Politecnico di Torino, 10129 Torino, Italy (e-mail: michela.meo@icar.cnr.it; leonardi@polito.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2010.2046745

techniques have been proposed to design "self-structured" P2P systems, so called because the association of descriptors to hosts is not predetermined, but adapts to the modification of the environment [9]–[11].

This paper, which is an extended and enhanced version of a paper presented at the IEEE International Symposium CCGrid 2009 [12], introduces *Self-Chord*, a P2P system that inherits from Chord the ability to construct and maintain a structured ring of peers, but features enhanced functionalities achieved through the activity of ant-inspired mobile agents. As opposed to Chord, Self-Chord decouples the naming of resources and peers, resulting in two sets of keys/indices that can have different cardinalities. Keys can be assigned to resources depending on the requirements of every specific application domain and on the desired granularity of resource categorization, with also the possibility to give them a semantic meaning. Moreover, Self-Chord does not place resource keys to specified hosts, as Chord does; this feature is actually unnecessary and limits the system flexibility. Conversely, Self-Chord focuses on the real objective, which is the reordering of keys over the ring and their fair distribution to the peers. Self-Chord agents move resource keys across the ring and sort them in a self-organizing fashion. The sorting of keys allows discovery operations to be executed in logarithmic time, exploiting the pointers (the *finger tables*) provided by the Chord structure.

The presented approach aims at opening a new research avenue for P2P frameworks because it combines the beneficial characteristics of structured P2P systems with methods from mobile agent and swarm intelligence technologies that offer significant self-organizing properties and so far have only been applied to unstructured systems. In particular, Self-Chord features the following benefits with respect to Chord.

1) In Self-Chord, there is no obligation to assign a key to a well-specified peer. This feature enables the definition of "classes" of resources—a class being defined as a set of resources that share common characteristics and are mapped to the same key value. A user can issue "class" queries, i.e., explore the network to find resources belonging to a specified class and then select the most appropriate for his/her purpose.
2) Structured systems like Chord can produce imbalance problems depending on the location of peers and the statistical distribution of the values of resource keys. In Self-Chord, the keys are fairly distributed over the peers, irrespective of the location of peers and the distribution of key values.
3) In Chord, appropriate operations are necessary when a peer joins the ring or when new resources are published: These resources must be immediately assigned to the peers whose indexes match the resource keys. These operations are not necessary in Self-Chord because the mobile agents are always active and will spontaneously reorganize the keys. This assures scalability (keys are continuously reordered as the network grows) and robustness with respect to environmental changes.

The main objective of this work is to show how the most popular structured system, Chord, can be enriched with self-organization and adaptive properties. However, the approach is easily generalizable, as discussed in Section VI: Similar bio-inspired algorithms can be defined for any other structured system, such as CAN [13] and Pastry [14].

The paper also aims to be a step toward the mathematical analysis of bio-inspired systems and swarm intelligence phenomena. The modeling of these systems through fluid-like differential equations has proved to be a valid approach. Indeed, the results obtained by solving these equations are comparable to those obtained with an event-based simulator and with a Java prototype available at the Web site http://self-chord.icar.cnr.it.

The rest of the paper is organized as follows. The following section gives an overview of Self-Chord and compares the results obtained with simulation and with the analytical model. Section III gives the details about the self-organizing technique that allows resource keys to be reordered and fairly distributed to the peers. Section IV shows the results of simulation experiments that confirm the effectiveness of Self-Chord in large networks, with particular emphasis given to the performance of discovery procedures and to important features such as scalability, load balancing, and dynamic behavior. Section V summarizes the benefits of the bio-inspired approach adopted by Self-Chord with respect to classical structured systems like Chord. Section VI discusses the state of the art in the fields of interest. Section VII concludes the paper.

II. Self-Chord

A. Basic Model

In Self-Chord, peers are organized in a logical ring. Each peer is given an index, having B_p bits, which is obtained with a uniform hash function and can have values between 0 and $2^{B_p}-1$. The ring is constructed and maintained as in Chord (see [6] for the details). Each resource is associated with a binary key, having B_c bits, which will be used to discover and access the resource. The number of possible values of the resource key, $N_c = 2^{B_c}$, can be viewed as the number of classes in which the resources are categorized. A *class* is defined as a set of resources having a specified set of characteristics and is therefore associated to the same value of the key. The values of resource keys can be obtained in two ways. The first is through the use of a *locality preserving* hash function [15], which assures that similar keys are associated to similar resources. Alternatively, resource keys can be given a semantic meaning: For example, the value of each bit may indicate the presence/absence of a specific topic [16], if the resource is a document.

In Chord, B_p and B_c must be set to the same value because there is a precise association between resources and peers. Conversely, in Self-Chord, the values of B_p and B_c can be set independently: The granularity of resource categorization may be chosen depending on the specific application domain, without any constraint related to the range of peer indexes. Consequently, there is no obligation to assign a key to the peer having the same index, or to its successor, as in Chord. However, to inherit the efficiency of resource discovery operations offered by Chord, the resource keys must be ordered on the ring. While in Chord, ordering is the outcome of a global planning, in Self-Chord it is obtained by the operations of ant-inspired agents that move the resource keys across the ring.

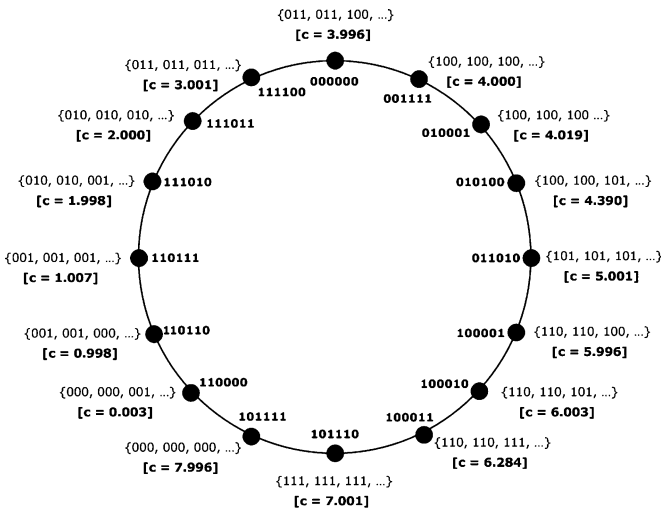


Fig. 1. Distribution and ordering of resource keys in the Self-Chord peers.

Ordering is performed through the concept of peer *centroid*. This is defined as a real value, between 0 and N_c, which minimizes the average distance between itself and all the keys stored by the current peer and by the two adjacent peers on the ring.[1] For example, with $N_c = 64$, a peer that stores three keys with values {4,6,8} (assuming for simplicity that the two adjacent peers do not store any key) has a centroid equal to 6. With another example, a peer that stores two keys with values {63, 0} has a centroid equal to 63.5. The value of the centroid is an indication about the keys stored in the local region of the ring and is used by agents to order keys. In fact, the agents tend to move each key to the peer whose centroid is as close as possible to that key.

Fig. 1 gives an example of the way resource keys are ordered on the ring. The values of B_p and B_c are respectively equal to 6 and 3. At the interior of the ring, the figure specifies the indexes of the peers, whereas at the exterior it reports, for each peer, the keys stored by the peer (only the first three keys are shown for simplicity) and the peer centroid c. It can be noted that the values of the centroids are ordered increasingly in clockwise direction as well as the keys stored in successive peers. Moreover, there is no relation between the resource keys and the peer indexes, owing to the self-organizing and "self-structured" approach used by Self-Chord. Indeed, the peer indexes are ordered by the Chord management operations, whereas the resource keys are ordered by agents in an independent fashion.

The agents do not operate forever, but are generated and die as the ants to which they are inspired. Each peer of the ring, at time of its connection to the network, generates a mobile agent with a given probability P_{gen}. The lifetime of this agent is randomly generated with a statistical distribution whose average is related to the average connection time of the connecting peer, calculated on the past activity of this peer. Therefore, the turnover rate and the average number of operating agents are related to the dynamic characteristics of the network, i.e., to the frequency of peer joinings and departures. Specifically, if the average connection time of all the peers is T_{peer}, and the average lifetime of agents is T_{agent}, the average number of agents $\bar{N}_a$ that circulate in the network at a given instant of time is associated with the average number of peers present in the network at a time, $\overline{N}_p$, in the following way:

$$\bar{N}_a \cong \bar{N}_p \cdot P_{\text{gen}} \cdot \frac{T_{\text{agent}}}{T_{\text{peer}}}. \tag{1}$$

This statistical relationship does not imply that any peer must be aware of the values of T_{peer} and $\bar{N}_p$. However, it can be used to tune the speed of the reordering process and the traffic load. For example, if each agent is given a lifetime equal to the average connection time of the peer that generates it, on average it results that $T_{\text{peer}} \cong T_{\text{agent}}$, therefore the average number of agents is P_{gen} times the average number of peers connected to the ring.

After a transient phase, the keys will be ordered on the ring, and the obtained order is robust with respect to successive modifications of the environment, for example to peers' connections and disconnections. This ordering allows Self-Chord to rapidly serve discovery requests, as a query can move across the ring toward the desired key. Discovery procedures exploit Chord-like *finger tables* in order to assure logarithmic discovery times, as will be explained in Section IV.

B. Operations of Self-Chord Agents

Each mobile agent gives its contribution to the reordering of resource keys on the ring. Two different approaches are discussed in the following. They will be referred to as "linear" and "logarithmic," respectively. We start by presenting and discussing the first approach.

According to the linear ordering, resource keys are always moved by agents between adjacent peers. Each agent periodically hops from a peer to its predecessor or successor, depending on the agent being left-handed or right-handed. When an agent, which currently does not carry any key, moves to a new peer, it must decide whether or not to take a key out of the visited peer. On the assumption that the centroid of the current peer is c and the agent is right-handed, the agent examines only the keys whose value r is higher than c because these keys should be moved to *successor* peers to improve the overall ordering. These keys are identified by evaluating the condition

$$c < r < (c + N_c/2) \bmod (N_c). \tag{2}$$

Conversely, a left-handed agent, which will move toward *predecessor* peers, evaluates the keys that satisfy

$$(c - N_c/2) \bmod (N_c) < r < c. \tag{3}$$

All calculations must take into account the circular ordering of peer and resource keys. To make the following discussion more fluent, this assumption will be given for granted. For example, conditions (2) and (3) can be simplified respectively to $c < r < c + N_c/2$ and $c - N_c/2 < r < c$.

To foster the correct ordering of keys, it is convenient to pick keys that are very different from the peer centroid, while keys that are similar to it are probably already placed in the correct place. Therefore, the probability of taking a key r is defined

[1]Key values are defined in a circular space, in which value 0 succeeds value $N_c - 1$: The distance between two values is defined as the length of the minimum circle segment that separates these values.

to be inversely proportional to the similarity between r and the peer centroid c. Accordingly, the similarity function $g_p(r,t)$ and the *take* probability $T_p(r,t)$ of resource r at time t in peer p are

$$g_p(r,t) = 1 - \frac{d_p(r,t)}{N_c/2} \tag{4}$$

$$T_p(r,t) = \frac{k_t}{k_t + g_p(r,t)} \quad \text{with } 0 \le k_t \le 1 \tag{5}$$

where $d_p(r,t)$ is the distance between r and the centroid of p, computed on the circular space of the keys. For example, with $N_c = 64$, the distance between 12 and 18.7 is equal to 6.7; the distance between 3 and 63.5 is 3.5. The value of $g_p(r,t)$ is comprised between 0 (maximum diversity between r and the centroid of p) and 1 (maximum similarity).

With high probability, the agent picks a key whose value is distant from the peer centroid. This key is carried by the agent and moved toward successor or predecessor peers, depending on the agent being right-handed or left-handed.

When, at time t, an agent that is carrying key r moves to a new peer p, it decides in a probabilistic fashion whether or not to leave the key on this peer. The *leave* probability, $L_p(r,t)$, is defined regardless the type of agent, left- or right-handed, and is

$$L_p(r,t) = \frac{g_p(r,t)}{k_l + g_p(r,t)} \quad \text{with } 0 \le k_l \le 1 \tag{6}$$

where r is the key carried by the agent and the similarity function $g_p(r,t)$ is computed as in expression (4).

As opposed to the take probability, the leave probability is directly proportional to the similarity between r and c, therefore the agent tends to leave a key if it is similar to the other keys stored in the local region of the ring. Clearly, this behavior contributes to the correct ordering of keys, which is also guaranteed by the fact that the centroid of a peer is calculated not only on the keys stored in the peer itself, but also on the keys stored by the two adjacent peers.

C. Steady-State Distribution of Keys

In order to evaluate the steady-state distribution of keys and centroids' position, we model the system evolution through a set of differential equations. The equations describe the dynamic evolution of the number of keys in the peers and are based on the assumption that the keys can be identified by continuous variables in $[0, N_c]$. Being N_c typically large, this assumption has a marginal impact on the results.

Let $f_p(x,t)$ denote the probability density function (pdf) of keys in peer p at time t. Also, let Λ be the arrival rate of an agent at a peer, and $\lambda_p^{(r)}(x,t)$ and $\lambda_p^{(l)}(x,t)$ denote the normalized arrival rate of agents carrying key x at time t in peer p, where (r) and (l) indicate the right-handed (clockwise) or left-handed (counterclockwise) rotation direction of the agent, respectively. The *take* probability for key x at time t in peer p are denoted by $T_p^{(r)}(x,t)$ and $T_p^{(l)}(x,t)$, again for right-handed and left-handed rotating agents, while the *leave* probability is indicated by $L_p(x,t)$ and does not depend on the agent rotation direction. Finally, $C_p(t)$ is the centroid of peer p at time t.

The equations describe the evolution of the arrival process of agents at the peers and of the distribution of keys in the peers.

Equations for λ and f: For the clockwise agent direction, we can write the following differential equation representing the evolution of the agent arrival rate in time:

$$\frac{\partial \lambda_p^{(r)}(x,t)}{\partial t} = \Lambda \left[(1 - L_p(x,t))\, \lambda_{p-1}^{(r)}(x,t) + T_p(x,t) f_p(x,t) \right]. \tag{7}$$

The first term in the equation describes the fact that an agent may leave peer p with key x if it was carrying x and x was not left in p; the second term represents the departure of an agent that takes x from p. Similarly, for the counterclockwise agent direction, we have

$$\frac{\partial \lambda_p^{(l)}(x,t)}{\partial t} = \Lambda \left[(1 - L_p(x,t))\, \lambda_{p+1}^{(l)}(x,t) + T_p(x,t) f_p(x,t) \right]. \tag{8}$$

The keys distribution is the combination of the arrivals of agents and take and leave probabilities according to the following equation:

$$\begin{aligned} \frac{\partial f_p(x,t)}{\partial t} = \Lambda \Big[& L_p(x,t) \lambda_p^{(r)}(x,t) - T_p^{(r)}(x,t) f_p(x,t) \\ & + L_p(x,t) \lambda_p^{(l)}(x,t) - T_p^{(l)}(x,t) f_p(x,t) \Big]. \end{aligned} \tag{9}$$

Computation of the Centroids: The value of peer p centroid depends on the distribution of the keys in this and in the two adjacent peers. $C_p(t)$ satisfies the following expression:

$$\begin{aligned} \frac{dC_p(t)}{dt} = & \frac{1}{3} \sum_{x=1}^{N_c} \Bigg\{ \frac{\partial d_p(x,t)}{\partial t} \left[f_p(x,t) + f_{s(p)}(x,t) + f_{p(p)}(x,t) \right] \\ & + d_p(x,t) \left[\frac{\partial f_p(x,t)}{\partial t} + \frac{\partial f_{s(p)}(x,t)}{\partial t} \right. \\ & \left. + \frac{\partial f_{p(p)}(x,t)}{\partial t} \right] \Bigg\} \end{aligned} \tag{10}$$

in which $d_p(x,t)$, as before, is the distance of x from the centroid of p at time t, and $s(p)$ and $p(p)$ denote, respectively, the successor and the predecessor peer of p.

The results obtained by solving the differential equations of the described model were compared to those of an event-based simulation. The simulator is derived from an object-oriented framework already adopted to evaluate the performance of a number of different distributed systems [9], [17]. In this version, objects are used to model the peers and the mobile agents that travel the network and perform the operations described in Section II-B.

We start by comparing the model and simulation results for a sample scenario in which $B_c = 6$ (resources are distributed among $N_c = 64$ classes), $B_p = 12$ (peer indexes are defined over 12 bits), and the number of peers actually connected in the ring is $N_p = 16$. The average number of resources published by a peer, referred to as N_{res}, is set to 10. Fig. 2 reports the centroid value of each peer with the model and the simulation in a steady situation. As expected, the centroid values are ordered and uniformly spaced, which proves the corresponding ordering of keys over the ring. Observe also that the model is

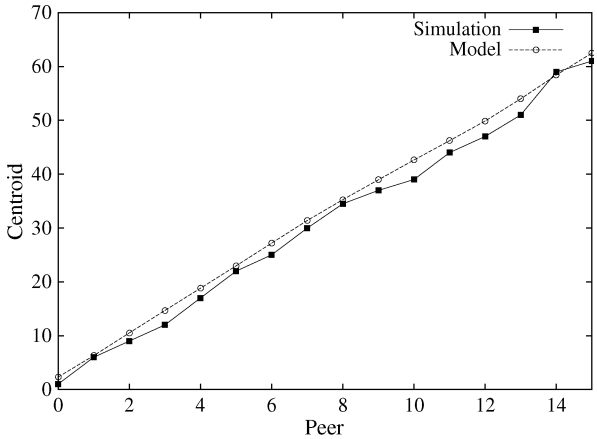


Fig. 2. Comparison between the analytical model and simulation results. Position of the centroids, with $N_p = 16$ and $N_c = 64$.

TABLE I
COMPARISON BETWEEN THE ANALYTICAL MODEL AND THE SIMULATION RESULTS: MEAN AND STANDAND DEVIATION OF THE DISTANCE BETWEEN KEYS AND CENTROID IN A PEER

		Mean dist. key-centroid		St. dev. dist. key-centroid	
N_p	N_c	Model	Simul.	Model	Simul.
16	16	0.945	1.143	0.846	1.024
16	64	3.770	4.142	3.520	3.849
16	128	7.545	7.802	7.053	7.491
64	128	1.984	2.042	1.865	1.943
128	128	1.012	1.031	1.006	1.029

very accurate, whereas simulation results show small fluctuations with respect to model values. To explain this, it must be considered that the model is a first-order approximation of the system dynamics.

The mean and standard deviation of the distance between the keys of a peer and the corresponding centroid were also evaluated. Table I reports the comparisons between simulation and analytical model for some scenarios with different values of N_p and N_c. Results are averaged over the peers. Again, the results are similar and prove that the keys stored in a peer are always very close to the value of the peer centroid, which is essential to guarantee an effective resource discovery, as will be shown in Section IV. Notice that the difference between simulation and model results decreases as the size of the system increases. In all the tested scenarios, with both model and simulation, the average distance between the centroid values of two consecutive peers is N_c/N_p, which confirms that the centroids are correctly ordered.

III. LOGARITHMIC ORDERING OF KEYS

To speed up the ordering phase, we propose the following modification to the agents movement. Agents move as described above (namely "linearly") only when they are not carrying any key, and in the same way, they perform the *take* operation. However, once an agent has taken a key from a peer, it moves directly to the region of the ring where this key should be deposited. In other words, it jumps toward the peer whose centroid is as close as possible to the carried key. To calculate the length of the jump, the agent exploits the fact that the peers are already ordered (based on the results of a hash function, as done in Chord) and the resource keys are also being ordered, even if these two orderings are not dependent on each other.

In detail, the agent first calculates the difference $r - c$ in the arithmetic modulo N_c, where r is the value of the carried key and c is the centroid of the current peer. Then, it makes a proportion between this distance, calculated in the space of resource keys, and the distance between the current peer P_s and the "destination" peer P_d, calculated in the space of peer indexes[2]

$$\frac{r-c}{N_c} = \frac{P_d - P_s}{N_r}. \tag{11}$$

Accordingly, the agent jumps to a peer whose index is as close as possible to

$$P_d = P_s + \frac{N_r}{N_c}(r-c). \tag{12}$$

To do this, the agent exploits the *finger table* of P_s. In Chord, the ith finger of peer p, denoted by $p.finger(i)$ contains the index of the first peer, d, that succeeds p by at least 2^{i-1} along the Chord ring clockwise, namely $d = successor(p + 2^{i-1})$, $i = 1 \ldots B_p$. The finger table is used by Chord to speed up the discovery requests and serve them in a logarithmic time since, at every jump of the search message, the search space is actually halved. Note that the cost of the finger-table creation is marginal as compared to the cost of key distribution.

Self-Chord uses a *bidirectional* finger table, in which fingers are also directed to the peers that precede the current peer in the counterclockwise direction. A *reverse* finger, denoted as $p.rev\_finger(i)$, points to the peer $\overline{d} = predecessor(p - 2^{i-1})$, $i = 1 \ldots B_p$. This reverse finger structure is symmetrical to that used by Chord and can be easily maintained at the cost of doubling the storage memory. A similar structure has been defined in [18] and analyzed to evaluate the performance improvement that can be obtained for the discovery operations.

The agent selects the peer of the finger table whose index is the closest to P_d and exploits the corresponding finger pointer to get to that peer. At this point, the agent evaluates the "leave" operation, in the same fashion as with the linear approach. If the key is actually deposited, the agent will start again to move in a linear fashion and hop to the successor or predecessor peer until it will take another key. If the leave operation is not performed, the agent will make another "logarithmic" jump, trying to approach the region of the ring where the carried key should be deposited.

The reason why the reverse finger table must be defined can now be easily explained. While in Chord it is always possible to choose a finger that points to a peer whose index is not greater than the target peer, this cannot be assured in Self-Chord, as the placement of keys over the ring is based on the statistical agent operations instead of a well-defined assignment pattern. If only the forward finger table were available, an agent that overcomes the target peer in the clockwise direction could not move backward, but would be obliged to perform another round trip in the

[2]In formula (11), N_r is the number of potential index values that can be assigned to a peer, and is equal to 2^{B_p}.

clockwise direction to return to the target peer. With a bidirectional finger table, a key can be moved in both directions, so this problem does not occur. Intuitively, the logarithmic approach is much faster than the linear one, as a key is moved through much longer and better directed hops. On the other hand, the linear approach allows a fairer load balance among peers to be achieved. This issue is discussed in the following.

A. Comparison Between Linear and Logarithmic Approaches

The linear and logarithmic approaches were evaluated with the event-based simulator for a scenario in which $B_c = 8$ (resources are distributed among $N_c = 256$ classes), $B_p = 12$ (peer indexes are defined over 12 bits), and the number of peers actually connected in the ring is $N_p = 256$. The average number of resources published by a peer, referred to as N_{res}, is set to 10. At the beginning, the resource keys are distributed randomly, then they are ordered through the operations of Self-Chord agents. In particular, each peer issues an agent, left-handed or right-handed with the same probability. The connection time of a single peer is distributed with a Gamma probability function. Each peer has a different average connection time, and the global average for all the peers, T_{peer}, is set to 5 h. The average lifetime of an agent is set to the average connection time of the peer that generates the agent. After receiving an agent, a peer forwards it to another peer, according to one of the policies defined above, after a random interval T_{mov}. Since the Self-Chord procedures can be accelerated or decelerated by tuning the value of T_{mov}, this parameter will be used as a time unit, and the performance results versus time will be reported accordingly. The parameters k_t and k_l, used in expressions (5) and (6), are respectively set to 0.3 and 0.9.

To evaluate the effectiveness of agent operations, we consider the distance (in the space of resource keys) from the centroids of every two consecutive peers and compute the mean and the standard deviation. In fact, when the keys are correctly ordered across the ring, the centroid values of the peers should be ordered and equally spaced, and the distance between any two consecutive centroids should always be comparable to N_c/N_p. Therefore, the average of this distance should be about N_c/N_p, and the standard deviation should go to zero.

Fig. 3 shows that, starting from a state with maximum disorder at time 0 and owing to agent operations, the mean of the centroid distance decreases from very high values to values that are almost equal to the theoretical value N_c/N_p, confirming the capacity of the Self-Chord algorithm to order the keys on the ring. However, the velocity of the reordering process is very different when using the linear or the logarithmic approach. With the latter, reordering is achieved after about 3000 time units. To obtain the value in seconds, the number of time units must be multiplied by the agent forwarding time T_{mov}. This result shows that Self-Chord is able to reorder the keys in an acceptable time even starting from a very unfortunate (and unrealistic) situation, in which all the peers join the system at the same time and, since resource keys are assigned with a uniform hash function, the disorder is maximum. In a real system, the peers join the ring gradually, and the new keys are positioned by agents among a large number of keys that are already correctly sorted, which is a much easier task. This gradual sorting process is much faster,

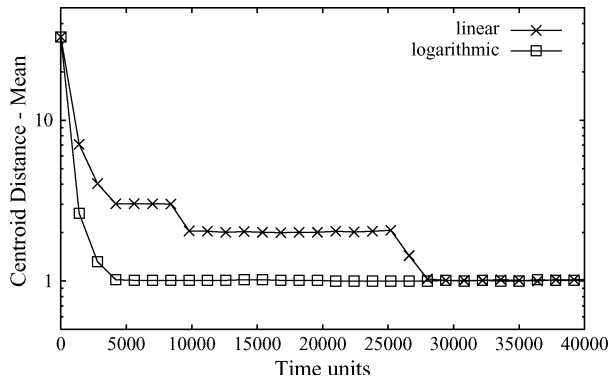


Fig. 3. Average distance between two consecutive centroids with linear and logarithmic approaches.

and a new key can be moved to the correct peer in logarithmic time. This issue will be better analyzed in Section IV-C. On the other hand, with the linear approach, the process takes about 30 000 time units, a time interval 10 times longer. Of course, the difference between the two approaches increases with the size of the system, i.e., with the number of peers and/or the overall number of keys that must be ordered.

The figure shows that the linear reordering process experiences periods in which the average centroid distance is stable, interleaved by abrupt changes. The reason for this behavior is that in the transient phase the keys are only partially ordered over the ring and the centroid values do not complete a single circle over the whole ring, but instead two or more circles. The reordering process progressively lowers the number of circles until it is reduced to 1, which corresponds to a correct and complete ordering. Each partial ordering with more than one circle creates an equilibrium state that the agents take some time to force, which explains the time intervals in which the average distance between consecutive centroids is almost constant. However, as the agents manage to achieve a transition from k circles to $k - 1$, an abrupt reduction in the average centroid distance is experienced.

Unfortunately, the logarithmic approach has an important drawback concerning the distribution of load among the peers. With the linear approach, each peer receives agents exclusively from the two adjacent peers, and therefore all the peers tend to store the same number of keys. Conversely, with the logarithmic approach, an agent that is carrying a key is obliged to use the peer finger table in its movements on the ring. However, it is known that the number of fingers that "point" to a given peer is not a constant, but depends on the indexes of this peer and its neighbors on the ring. In fact, peer indexes are not uniformly distributed over the ring: If a range of admissible index values is considered, the number of connected peers whose indexes are comprised in this range has a logarithmic distribution [6]. A peer is pointed by a large number of fingers if it is the successor (or predecessor) of a large number of indexes—in other words, if it is the first peer after a long range of indexes that are not assigned to any peer in the network. This imbalance in the number of inward fingers causes a corresponding imbalance in the number of agents delivered to the peers through the fingers and—since each of these agents carries a key that may be

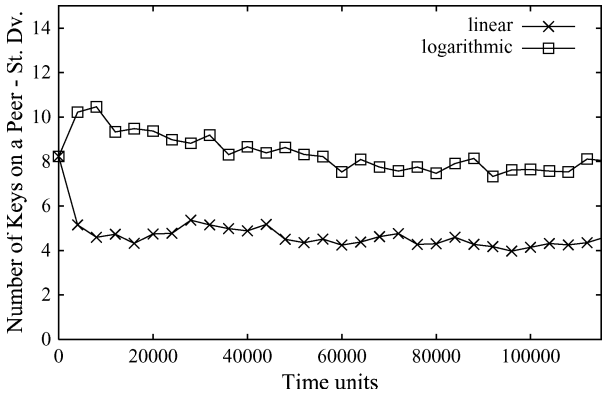


Fig. 4. Standard deviation of the number of keys stored by a peer with linear and logarithmic approaches.

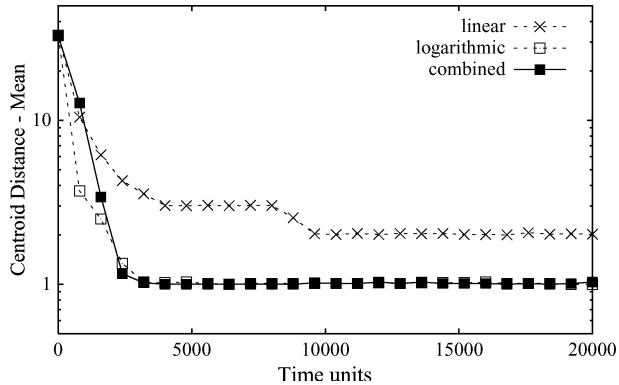


Fig. 5. Average distance between two consecutive centroids with linear, logarithmic, and combined approach.

deposited with a leave operation—in the number of keys that are stored by peers.

This is confirmed by the results in Fig. 4, which show the standard deviation of the number of keys stored by a peer (the mean of this variable is equal to 10, the average number of resources published by a peer). In a steady situation, the value of this index fluctuates around a value of 4.0 in the case of linear ordering, whereas it approximately doubles with logarithmic ordering. This confirms that the logarithmic ordering does not guarantee a good load balance among the peers.

B. Switch Between Logarithmic and Linear Approach

The best solution seems to be an approach that combines the rapidity of logarithmic ordering with the fair load distribution assured by linear ordering. This can be achieved by starting the ordering process in the "logarithmic mode" in order to rapidly reorder the resource keys and, after the largest part of the reordering process has been performed, switching it to the "linear mode" to better distribute the keys among the peers.

The switch between the two modes must be performed by every peer only on the base of local information. Each peer knows the value of N_c, but in general does not know the value of N_p, the number of peers of the network. Therefore, a peer cannot base its decision on the average distance between consecutive centroids in a local sector of the ring and on its proximity to the theoretical value N_c/N_p. However, it is observed that this value decreases with time, with a slope that is high at the beginning and then lower and lower as the curve approaches the mentioned value of N_c/N_p, as shown in Fig. 3. Therefore, the derivative of the centroid distance can be used to perform the switch locally: The analysis of the derivative can be done without any knowledge on the system global state.

To estimate this derivative, each peer maintains a variable Δ that is updated every time interval—in our tests every 5 min—as follows:

$$\delta(i) = \frac{C_{\text{dist}}(i) - C_{\text{dist}}(i-1)}{N_c} \tag{13}$$

$$\Delta(0) = D_0 \tag{14}$$

$$\Delta(i) = \delta(i) + F_{\text{ev}} \cdot \Delta(i-1). \tag{15}$$

The term defined in expression (13) is the difference between the current and the last value of the average distance between consecutive centroids, evaluated in a local sector of the ring. The considered centroids are the centroid of the peer itself and those of a small number of neighbor peers in the two directions. The difference is then normalized over the number of resource classes N_c. Since the derivative of this index is generally negative (the average distance between consecutive centroids decreases), the initial value of Δ is set to a negative value, D_0. For successive calculations of Δ, the contribution of the past values is weighed through the evaporation factor F_{ev}. The switch from the logarithmic mode to the linear mode is performed as the value of Δ exceeds a given threshold T_h that is close to zero. The fact that the threshold is exceeded is an indication that the derivative is approaching a null value and that the average centroid distance is getting stable. This means that the ordering process is nearly completed and it is convenient to pass to the linear mode in order to better distribute the keys among the peers of the system.

The choice of the algorithm mode is performed as follows: Each peer initially sets its mode to "logarithmic," then it switches to "linear" as soon as the mentioned threshold for Δ is exceeded. When an agent carries a key, it moves according to the mode that is set on the current peer until it drops the key. As the ordering process proceeds, the peers will gradually pass to the linear mode, and so will the agents.

Figs. 5 and 6 compare the average centroid distance and the standard deviation of the number of keys per peer over the whole network and compare the results obtained with the logarithmic, the linear, and the combined process described so far. It is noticed that the switch mechanism accomplishes its purpose since the reordering process is nearly as fast as with the logarithmic mode and, as the peers switch from logarithmic to linear, the load is balanced as if the linear mode were used from the beginning. For this test, the parameters for the calculus of the derivative were set as follows: $D_0 = -0.1$, $F_{\text{ev}} = 0.9$, and $T_h = -0.01$. The local centroid distance was calculated over a local sector of seven peers. It is worth noting that the setting of these values is not very sensitive: The only effect that can derive from a small variation of one or several of these is that the switch of peers can be anticipated or delayed. This can reduce or extend the transient phase, but has hardly any effect on the behavior of the system in the steady situation.

To better illustrate the switch process, Fig. 7 shows the trend of the Δ parameter, averaged over all the peers, and in parallel

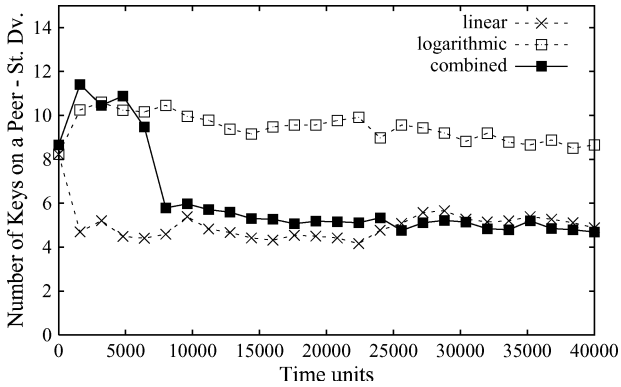


Fig. 6. Standard deviation of the number of keys stored by a peer, with linear, logarithmic, and combined approach.

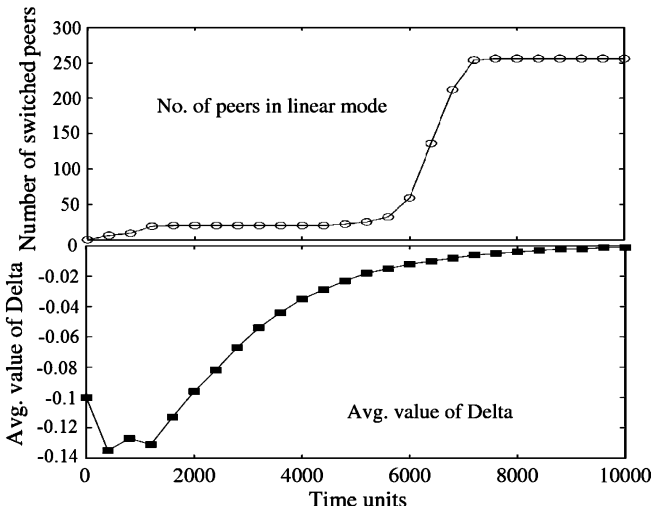


Fig. 7. Analysis of the combined approach. Parallel trend of the number of peers that have switched to the linear mode and of the value of the Δ parameter, averaged on all the peers.

the number of peers that have already switched to the linear mode. It is noticed that the largest part of peers operates its switch when the average value of Δ is around the threshold value, that is, -0.01.

An important issue to consider is how to manage new resource keys, for example those published by the peers that join the network. To speed up the correct placement of new keys, the agents that carry them use the logarithmic mode for a limited amount of time, irrespective of the mode of the peers that they traverse. This techniques will be better discussed in Section IV-C.

IV. Performance Analysis of Self-Chord

So far, all the tests were performed with a limited number of peers ($N_p = 256$) because the linear approach is very slow with a very large number of peers. In this section, however, the combined approach is adopted for all the tests, with the mode switch mechanism described in Section III-B. This allows results to be obtained with larger and more realistic networks.

In the tests, the number of resource classes N_c is set to 1024, which corresponds to a number of bits in resource keys, B_c, equal to 10. The parameters P_{gen}, N_{res}, and T_{peer} are set as in

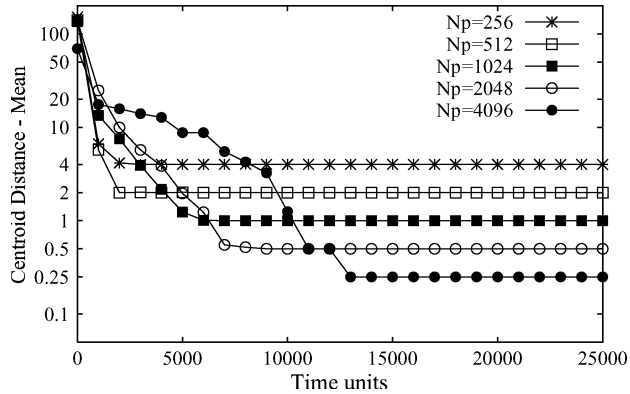


Fig. 8. Average distance between two consecutive centroids with 1024 resource classes and variable number of peers.

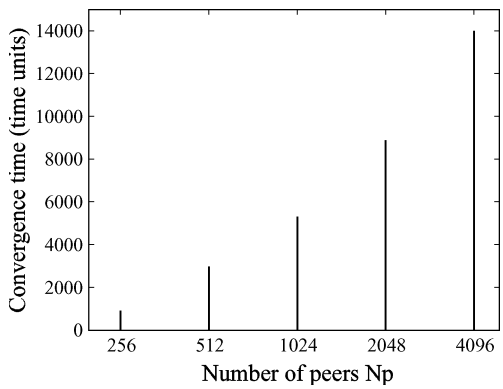


Fig. 9. Convergence time for different network sizes.

Section III-A: Their respective values are 1.0, 10 resources per peer, and 5 h. The parameters k_t and k_l are set to 0.3 and 0.9. The average lifetime of an agent is set to the average connection time of the peer that generates the agent: From formula (1), the number of agents that travel the network is on average equal to the number of connected peers. Note that these settings do not influence the behavior of Self-Chord in the steady situation, but can only affect the transient phase: For example, a larger number of agents would reduce the duration of the transient phase.

A. Scalability Analysis

This section presents simulation results obtained with a variable number of peers N_p ranging from 256 to 4096. Fig. 8 shows the trend of the average distance between consecutive centroids. In all the experiments, reordering is performed correctly, which is confirmed by the fact that this index tends to N_c/N_p, the value corresponding to an equally spaced ordering of centroids. Of course, the time needed to reorder the keys increases with the number of peers. This is better shown in Fig. 9, which reports the trend of the convergence time, defined as the time at which the average centroid distance falls within the range $N_c/N_p \pm 5\%$, with respect to the number of peers. The values in Fig. 9 are obtained by taking the average from 20 simulation runs.

Fig. 10 reports the distribution of the relative distance between the keys and the centroids of the peers on which these keys are located in a network with 4096 peers in a steady condition. This figure shows that the value of the large majority of

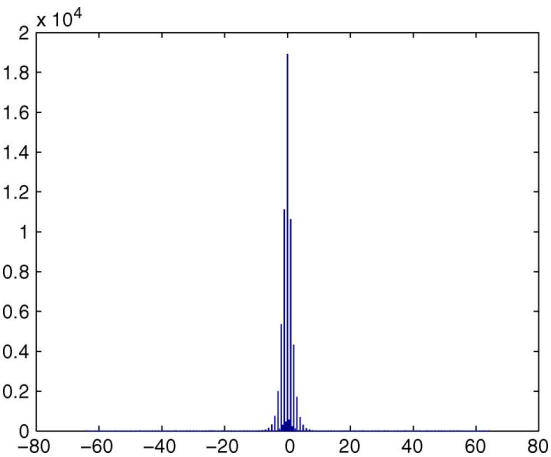


Fig. 10. Distribution of keys with respect to the peer centroid.

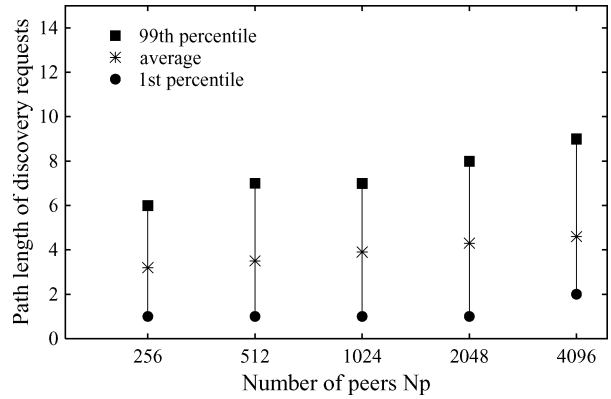


Fig. 11. Path length of discovery requests: average, 1st, and 99th percentile.

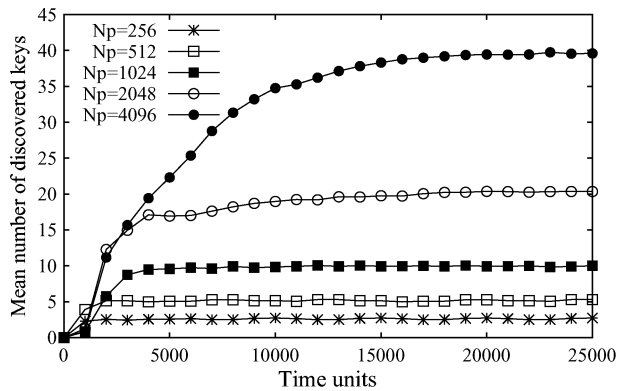


Fig. 12. Average number of target descriptors discovered by a query versus time, with variable number of peers.

the keys is very close to the peer centroid: The number of possible key values is $N_c = 1024 =$, but the difference between the value of a key and that of the respective centroid is very rarely larger than 5. This means that a discovery procedure, issued to search the keys with a specific value, can be very effective if the query message is directed to a peer in which the centroid value is as close as possible to the target key value. In fact, almost all the keys having the desired value can be found on this and in a few adjacent peers.

Performance of Discovery Requests: The ordering of keys over the Self-Chord ring is profitably exploited by the discovery procedure: At each step, the query message is forwarded through the finger tables to the peer whose centroid is estimated to be the closest to the target key value. As with the logarithmic ordering approach, the destination peer is selected by making a proportion between the resource keys and the peer indexes [see expressions (11) and (12)]. If the centroid of the destination peer is found to be closer to the target key than to the centroid of the current peer, the query message is forwarded to the destination peer, and the discovery procedure continues. Whenever this condition is not satisfied, or the destination peer coincides with the current peer, the discovery procedure terminates.

The number of steps needed to reach the target peer is logarithmic with respect to the number of peers since each step allows the search space to be approximately halved, as in Chord [6]. Fig. 11 reports the average, the 1st and the 99th percentile of the path length, defined as the number of steps/jumps performed by a search message. Here, it is worth recalling that the average number of steps experienced in Chord is equal to $(1/2)\lg_2 N_p$ [6], but it is reduced to $(1/3)\lg_2 N_p$ in [18] in consequence of the presence of the reverse finger table. Fig. 11 shows that, also in Self-Chord, the average number of steps is always very close to $(1/3)\lg_2 N_p$. Moreover, the 99th percentile is lower than $\lg_2 N_p$, meaning that the search process is very fast also in the most unfortunate cases.

Fig. 12 shows the mean number of keys discovered by a search request for different values of N_p. The assumption is that a search message, after completing its path, retrieves all the keys having the desired value that are located on the current peer and on four adjacent peers, two on the left and two on the right. Indeed, owing to the statistical nature of the reordering process, it is possible also that these neighbor peers store a significant number of keys having the desired value. In Fig. 12, the number of discovered keys is reported versus time: It can be observed that this index gradually increases as the agents relocate the keys. It is also noticed that the steady value is comparable to $(N_p * N_{\text{res}})/N_c$. In fact, this is the average number of keys of a specific class that are published in a network having N_p peers, in the case that N_{res} is the average number of resources published by a peer (10 in these experiments) and N_c is the number of resource classes (1024). In conclusion, the discovery procedure successfully discovers nearly all the keys that have the desired value.

B. Nonuniform Distribution of Keys

So far, the performance of Self-Chord has been analyzed under the assumption that the values of the keys associated with the resources are distributed uniformly. This assumption is generally valid in the case that the keys are computed with a hash function, but still there can be very popular resources that map to the same key. Moreover, in Self-Chord, a resource key can have a semantic meaning: For example, if the resource is a document, a bit of the key can express the fact that a document focuses or not on a given topic. In a case like this, some key values can be more frequent than others. The self-organization of keys performed by Self-Chord agents allows the load to be

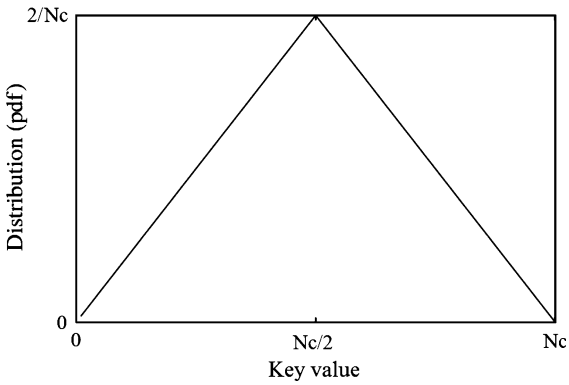


Fig. 13. Example of a nonuniform distribution of keys: the triangular distribution. With the number of resource classes set to N_c, $N_c/2$ is the most frequent key value.

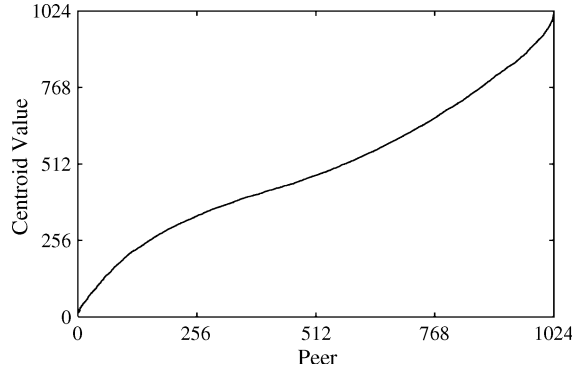


Fig. 14. Centroid values of peers. The first peer reported in the x-axis is the one with the lowest centroid value. The others are taken from the ring following the clockwise direction.

fairly balanced among the peers, even in the case of nonuniform distribution.

A set of experiments was performed assuming that the key values are distributed with a triangular distribution. More specifically, if the number of admissible key values is N_c, it is assumed that $N_c/2$ is the most frequent value, whereas values 0 and $N_c - 1$ are the least frequent. Fig. 13 shows the pdf of the triangular distribution that is obtained with these assumptions.

In classical structured P2P systems, a nonuniform distribution of keys produces a nonuniform balance of load. In Chord, for example, under the described triangular distribution, the peer with index $N_c/2$ would store a large number of keys since it would be assigned the keys of the most popular resources. Conversely, Self-Chord distributes the keys to the peers in a fair fashion, with every distribution of keys. In the case of nonuniform distribution, the most popular keys are placed by the agents on several adjacent peers so that no peer is given the responsibility of storing a large number of keys. The consequence is that the centroids of the peers that store the most popular keys are close to each other since the stored keys are similar. This phenomenon is shown in Fig. 14, which reports the centroid values of all the peers, in a network in which N_p and N_c are both equal to 1024. The first peer on the x-axis is the one that has the lowest centroid value, and the other peers are taken from the ring following the clockwise direction. The trend of the figure confirms that many peers have centroid values that are close to the most frequent key, $N_c/2$, in this case equal to 512, while fewer peers have centroids with values close to infrequent keys.

The distribution of the number of keys stored in a peer confirms the fair balance of load. The average, the 1st, and the 99th percentile of this index were found to have the same values with both the uniform and the triangular distribution of keys and are equal to 10, 2, and 22, respectively. The improvement versus Chord is considerable. For example, the 99th percentile calculated in Chord under the uniform assumption and reported in [6] is about 50, compared to the value of 22 experienced in Self-Chord. With a nonuniform distribution, an acceptable load balance can be maintained in Chord only by defining additional structures, specifically with the use of a number of virtual nodes on each real peer. Self-Chord does not need any superstructure to achieve a fair load balance.

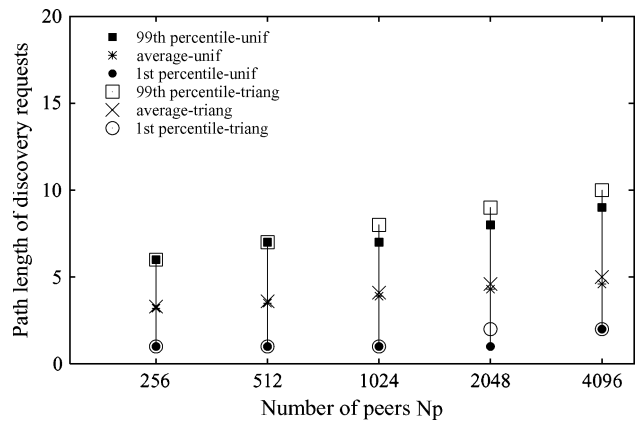


Fig. 15. Path length of discovery requests: average, 1st, and 99th percentile calculated with uniform and triangular distributions of keys and a variable value of N_p.

In Section IV-A, it was mentioned that the resource discovery algorithm estimates the index of the destination peer to which a query message is forwarded. However, if the distribution of keys is not uniform, the destination peer could have a different centroid value than the estimated one. Therefore, a set of experiments was run to assess the performance of the discovery procedure in the case that the key distribution is triangular. Fig. 15 reports the average, 1st, and 99th percentile of the number of steps made by search messages and compares the values obtained with the uniform and the triangular distributions of keys. The comparison shows that an inaccurate estimation of the centroid value of the destination peer, due to nonuniform distribution of keys, is rapidly compensated by a few more steps of the search message: The path length of search messages is still logarithmic with respect to the number of peers.

C. Dynamic Behavior

Self-Chord has an important advantage versus Chord also in terms of network and processing load. In a structured system like Chord, the keys of new resources—for example, those published by new or reconnecting peers—must be immediately placed in specified hosts; this can originate a high load if many resources are published in a short interval of time. In Self-Chord, the load is invariant because a new peer does not

need to perform any additional operation (besides the operations related to the overlay management, such as the finger table creation). The keys of the new resources will be picked by the agents that pass by this peer. The processing load Λ can be defined as the average number of agents per second that arrive and are processed at a peer. Λ can be calculated by multiplying the average number of agents $\overline{N}_a$ by the frequency of their movements $1/T_{\mathrm{mov}}$, so obtaining the number of times per second that an agent arrives at any peer, and then dividing the result by the average number of peers $\overline{N}_p$ to get the number of times per second that an agent arrives at a specific peer

$$\Lambda = \frac{\overline{N}_a}{\overline{N}_p \cdot T_{\mathrm{mov}}} \approx \frac{P_{\mathrm{gen}}}{T_{\mathrm{mov}}}. \tag{16}$$

The simplification is given by applying expression (1) in the case that T_{agent} is approximately equal to T_{peer}.

For example, if the average value of T_{mov} is equal to 5 s, and P_{gen} is set to 1.0, each peer receives and processes about one agent every 5 s, which is an acceptable load since take and leave operations are very simple. This result, obtained theoretically, was confirmed by simulation. Note that the processing load does not depend on the frequency of peer joinings and disconnections, the network size, nor the average number of resources published by a node, which confirms the scalability properties of Self-Chord.

To speed up the correct displacement of the keys of new resources, they are moved by agents using the logarithmic approach. Specifically, an agent that picks a "new" key uses the finger table to jump to the next peer, even if the current peer has already switched to the "linear" mode (see Section III-B). In fact, the linear mode would oblige the agent to carry the key through all the intermediate peers before depositing it in the correct peer. This functionality is easily obtained: A counter is set on every new key, and it is used to make the agents move the key with the logarithmic mode for a fixed number of times. To assure that the key is placed in the correct peer in logarithmic time, it suffices to set the counter to a value equal or larger than $\log(N_p)$. Of course, if the value of N_p is not known, which is the general case, it can be overestimated by the peers in order to set the counter to an appropriate value.

The results discussed so far showed that the Self-Chord agents can reorder the keys starting from a completely disordered network. Normal circumstances are much less stressful: If the network grows gradually, the correct sorting of the keys can be kept with a few agent operations that move the new keys to the correct place of the ring. However, a set of specific experiments was performed to evaluate the dynamic behavior of Self-Chord in more disadvantageous situations: Once the reordering process has reached a steady condition, a perturbation is generated by simulating the simultaneous arrival of a large number of new peers, each with 10 new resources on average. The initial number of peers N_p is set to 1024, but after 10 000 time units, a number of new peers, specified as a percentage P_{join} of N_p, join the network.

Performance analysis focuses on the average distance between consecutive centroids since this index gives an immediate indication about the effective reordering of keys over the network. Fig. 16 shows the value of this index before and after the perturbation induced by the joining of a percentage P_{join} of new peers, with P_{join} set to 25%, 50%, and 100%, corresponding respectively to 256, 512, and 1024 peers. The index experiences a sudden and prominent increase at the joining time: Since the new keys are published randomly by the peers, the key ordering is disturbed. However, the agents replace the new keys and restore the correct ordering very rapidly, in a time ranging from 40 to 200 time units. It can be noticed that the steady value of the average centroid distance, after the perturbation, becomes equal to the new value of N_c/N_p. With N_c always set to 1024, the value of the ratio is equal, in the three examined cases, to 4/5, 2/3, and 1/2, respectively. The comparison between Figs. 16 and 8 is interesting. While the agents take about 5000 time units to order the keys in a network with 1024 peers, if they start from scratch, they take only 200 time units to order the keys published by additional 1024 peers. This confirms that the insertion of new keys in an ordered network is an easy task and proves that Self-Chord is naturally scalable if peers join the network gradually, which is the expected behavior in a real network. In a steady condition, any perturbation, even very intense, such as those considered in Fig. 16, is easily managed by Self-Chord agents, and the key ordering is recovered rapidly.

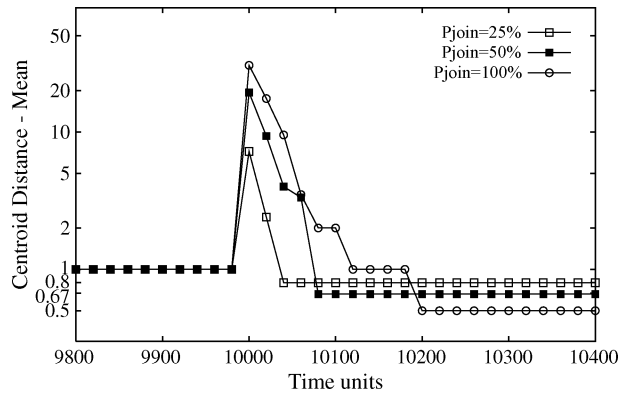


Fig. 16. Average distance between two consecutive centroids. At the beginning, the values of N_p and N_c are set to 1024. After 10 000 time units, a percentage P_{join} of new peers join the network.

The disconnection of a peer is very simple to manage. In Chord, the keys are consigned to the successor peer because this is the peer devoted to handle them. In Self-Chord, they are passed half to the successor and half to the predecessor peer, thus improving the load balance even in this respect.

V. Discussion

This paper presents Self-Chord, a "self-structured" P2P system built according to a bio-inspired algorithm. The reported results allow for a comprehensive analysis of Self-Chord and its comparison to Chord. The most important functionality of Chord, logarithmic discovery time, is preserved. In addition, due to the self-organizing activities of agents, Self-Chord features several benefits, as illustrated in the following.

1) *Better support of complex discovery requests*. In Chord, peer and resource keys are defined on the same number of bits by hash functions. This forces users to search for a specific resource, using its key as a search parameter, and hinders the execution of more complex discovery

requests. In Self-Chord, the definition of resource keys is flexible and uncorrelated with peer indexes, and it is also possible to give a semantic meaning to key values. This enables the system to serve "class" queries, issued to search for resources having common characteristics. A user can explore the network to find a number of resources belonging to the same class, and then select the most appropriate for its purpose. This is a typical problem in Grid and Cloud computing. For example, a user might search for hosts for which the CPU speed and the memory size are within a specified range, and choose among the discovered results in a successive phase.

2) *Better balance of storage load*. Self-Chord improves the balance of storage load among peers. In Chord, a peer is responsible for all the keys whose values are between its index and the index of the predecessor peer on the ring. Therefore, a peer might store a large number of keys if the distance between this peer and its predecessor is large. Moreover, if some resources are more popular than others, imbalance problems are even worse because the peers that store popular keys may be overloaded. In Self-Chord, the number of keys stored by a peer neither depends on the distance from its predecessor nor on the popularity distribution of keys. The work of agents in Self-Chord is capable of significantly improving the load balance, with respect to Chord, even with a uniform distribution of keys, and the advantage increases with a nonuniform distribution.
3) *Improved dynamic behavior*. In Chord, appropriate operations are necessary when a peer joins the ring or when new resources are published. These resources must be immediately assigned to the peers whose indexes match the resource keys. Therefore, the computational load depends on the dynamic behavior of the system, specifically on the churn rate of peers. Conversely, in Self-Chord the computational load is constant because the mobile agents continuously reorganize the keys. Any perturbation of the steady condition, even very intense, is efficiently managed, and the key ordering is recovered rapidly. Indeed, the placement of new/modified keys in the correct position of the ring is done in a logarithmic time, so it is as fast as a resource discovery operation. The system is also robust with respect to modifications of resource properties: If the value of a resource key changes, due to a modification of the resource, the key will soon be moved by the agents that, by recognizing that the key has become an outlier in the current peer, will assign a large take probability to it.

It should be remarked here that all these improvements are obtained in a totally decentralized and self-organizing fashion, while they would be very difficult to achieve with any centralized algorithm. This confirms the surprising efficacy of these very simple nature-inspired mechanisms, especially when they are adopted in a large distributed environment.

VI. Related Work

In most distributed systems, information services are implemented in accordance with centralized or hierarchical architectures, mostly because the client/server approach is still used today in the majority of distributed systems and in service-oriented frameworks. However, these architectures are impractical when a large number of resources, not all of which are under the control of the same organization, must be coordinated, as in the case of multi-institutional Grids or Clouds [1], [3]. Major drawbacks are poor scalability, limited autonomy of organizations, unfair balance of load, lack of fault-tolerance owing to the presence of single points of failure, or bottlenecks [17]. In the last few years, the P2P paradigm has emerged as an alternative to centralized and hierarchical architectures. Novel approaches for the construction of scalable and efficient information systems need to have the following properties [19], [20]: self-organization (meaning that components are autonomous and do not rely on any external supervisor), decentralization (decisions are to be taken only on the basis of local information), and adaptivity (mechanisms must be provided to cope with the dynamic characteristics of hosts and resources).

Requirements and properties of "Self-Organizing Grids" are sketched in [7]. In the architecture proposed in [21], Grid nodes self-organize in groups on the basis of the similarity among the resources that they offer to the network. Each group elects a leader node that receives requests tailored to the discovery of resources that are likely to be maintained by the group. This is an interesting approach, but it still has nonscalable characteristics. For example, it is required that each node has a link to all the leader nodes, which is problematic in a very large system. A self-organizing mechanism is also exploited in [22] to build an adaptive overlay structure for the execution of a large number of tasks in a Grid.

The Self-Chord algorithm presented in this paper shares several characteristics from mobile agent systems (MAS), which are often adopted to emulate the behavior of biological systems [23]. For example, insects and birds can be imitated by mobile agents that travel through the hosts of a distributed system and perform their simple operations. Agent-based systems may inherit useful and beneficial properties from biological counterparts, such as self-organization, decentralization, and adaptivity. Coordination among agents is essential to improve the effectiveness of their tasks, in particular for resource discovery. Coordination is usually achieved through a direct exchange of messages among agents. Conversely, the Self-Chord system presented here exploits the *stigmergy* paradigm [24]: Agents interact and cooperate through the modifications of the environment that are induced by their operations. In fact, the behavior of an agent is driven by the state of the local region of the system, which in turn is modified by the operations of other agents.

Self-Chord is specifically inspired by ant algorithms, a class of agent systems that can solve very complex problems by imitating the behavior of different species of ants [8]. Ant algorithms are one of the most popular examples of "swarm intelligence" systems, in which a number of agents follow very simple rules with no centralized control, and complex global behavior emerges from their local interactions. Among such systems, Anthill [25] is tailored to the design, implementation, and evaluation of P2P applications based on multiagent and evolutionary programming. It is composed of a collection of interconnected

nests. Each nest is a peer entity that makes its storage and computational resources available to swarms of *ants*, mobile agents that travel the network to satisfy user requests. Recently, ant algorithms have been proposed to design "self-structured" P2P systems, in which the association of keys with hosts is not predetermined, but adapts to the modification of the environment. In So-Grid [9], Grid resources are assumed to be precategorized in classes, and their descriptors are spatially clustered by ant-inspired mobile agents, thus facilitating the discovery of a cluster containing a large number of resources that belong to the desired class. Antares [11] extends this concept by using a locality-preserving hash function, which guarantees that similar resources are assigned similar key values. Keys are spatially sorted by mobile agents according to their key values. In this way, a search message can be driven toward the desired descriptors by following the gradient of resource keys. At each step, the message is forwarded to the neighbor peer that minimizes the distance between the keys stored there and the target key.

Anthill, So-Grid, and Antares are all unstructured P2P systems. Indeed, structured P2P systems have always been considered incompatible with self-organizing properties and adaptive behaviors. Self-Chord confutes this belief and proves to be both scalable and robust with respect to environmental modifications while retaining the benefits of structured P2P systems—in particular, logarithmic search time.

Another important objective of Self-Chord, as discussed in Section V, is the execution of class and range queries. The efficient execution of complex queries is indeed a very tough issue for distributed systems [26]. Some types of structured P2P systems are capable of serving class queries, but often at the cost of either maintaining complex tree-like structures [27] or increasing the traffic load by issuing a number of subqueries [28]. The Self-Chord information system naturally supports class and range queries. This feature derives from the flexibility provided by Self-Chord in the definition of the resources keys and their assignment to resources, and from the utilization of two separate algorithms that are used to reorder the peers and the keys in an independent fashion.

While the presented system is partly based on Chord, similar algorithms can be defined for any other structured system. For example, in CAN [13], resource keys are placed in a toroidal multidimensional structure: The position of a key over each dimension is equal to the value of a corresponding numerical parameter. An ant-inspired algorithm can be devised also in this case: A centroid can be defined to represent the keys stored in a restricted region of the multidimensional space. The keys will be moved by agents through adjacent peers by comparing their values to the values of the peer centroids, with the objective of sorting the keys over the multidimensional structure. The sorting of keys will enable the efficient execution of queries without requiring a rigid association among resource keys and peer codes.

Though the use of nature-inspired and "swarm intelligence" algorithms for distributed systems has notably increased in the last few years, there is a significant need for rigorous methodologies both for the design and the analysis of such algorithms. The difficulty of the design phase lies in the process of deriving a distributed protocol from a natural phenomenon. Existing approaches often tend to be informal, and thus the relation between phenomenon and protocol is often not quantifiable. In [10], the authors present a methodology that helps to model natural phenomena as difference equations and translate them into the so called "sequence protocols." The analysis issue is also nontrivial: The dynamics of collective behaviors are intrinsically stochastic and discrete, and so far they have been approximated by nonlinear differential equations only in a few cases. In [29], an analytical model is adopted to approximate the collective behavior of ants that, while searching for a path from their nest to a food source, must face a choice between two bridges that lead to the same food source. However, this very simple case is not easily generalizable. Nonlinear differential equations are also used in [30] to model the behavior of a swarm of robots that need to collaborate to perform an activity that would be impossible for a single robot—for example, pull and transport a heavy stick. The analytical model presented in this paper aims to be a step toward the mathematical analysis of nontrivial bio-inspired systems and swarm intelligence phenomena.

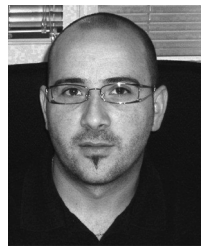
VII. Conclusion

This paper aims to open a new research avenue for P2P frameworks because it presents a P2P system that inherits the beneficial characteristics of structured systems, but offers further profitable characteristics inherited by biological systems, such as self-organization, adaptivity, scalability, and fast recovery from external perturbations. In Self-Chord, a set of ant-inspired mobile agents move and reorder the resource keys in a ring of peers in a self-organizing fashion without any predetermined association between keys and peers. Self-Chord efficiency and efficacy were confirmed by simulation results. In this paper, the presented ant-inspired approach is applied to Chord, but it could similarly be applied to other structured P2P systems in which peers are not organized in a ring, but in other structures such as multidimensional grids or trees. The paper also introduces an analytical framework that models the system evolution through a set of differential equations, representing a significant step toward the mathematical analysis of nontrivial bio-inspired systems and swarm intelligence phenomena. A Java prototype of Self-Chord is available at the Web site http://self-chord.icar.cnr.it.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA: Morgan Kaufmann, 2003.
[2] I. J. Taylor, *From P2P to Web Services and Grids: Peers in a Client/Server World*. New York: Springer, 2004.
[3] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008.
[4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. GCE*, Austin, TX, Nov. 2008, pp. 1–10.
[5] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surveys*, vol. 36, no. 4, pp. 335–371, 2004.
[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. ACM SIGCOMM*, San Diego, CA, 2001, pp. 149–160.
[7] D. C. Erdil, M. J. Lewis, and N. Abu-Ghazaleh, "Adaptive approach to information dissemination in self-organizing grids," in *Proc. ICAS*, Silicon Valley, CA, Jul. 2005, p. 55.
[8] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford Univ. Press, 1999.

[9] A. Forestiero, C. Mastroianni, and G. Spezzano, "So-Grid: A self-organizing grid featuring bio-inspired algorithms," *ACM Trans. Autonom. Adaptive Syst.*, vol. 3, no. 2, May 2008, Article no. 5.
[10] S. Y. Ko, I. Gupta, and Y. Jo, "A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing," *ACM Trans. Autonom. Adaptive Syst.*, vol. 3, no. 3, Aug. 2008, Article no. 11.
[11] A. Forestiero and C. Mastroianni, "A swarm algorithm for a self-structured P2P information system," *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 681–694, Aug. 2009.
[12] A. Forestiero, C. Mastroianni, and M. Meo, "Self-chord: A bio-inspired algorithm for structured P2P systems," in *Proc. 9th IEEE CCGrid*, May 2009, pp. 44–51.
[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, San Diego, CA, 2001, pp. 161–172.
[14] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. Middleware*, London, U.K., 2001, pp. 329–350.
[15] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," in *Proc. 4th GRID*, 2003, p. 184.
[16] C. Platzer and S. Dustdar, "A vector space search engine for Web services," in *Proc. 3rd ECOWS*, 2005, p. 62.
[17] C. Mastroianni, D. Talia, and O. Verta, "Designing an information system for grids: Comparing hierarchical, decentralized P2P and super-peer models," *Parallel Comput.*, vol. 34, no. 10, pp. 593–611, Oct. 2008.
[18] J. Jiang, R. Pan, C. Liang, and W. Wang, "Bichord: An improved approach for lookup routing in chord," in *Proc. 9th ADBIS*, Tallinn, Estonia, Sep. 2005, pp. 338–348.
[19] A. Iamnitchi, I. Foster, J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, "A peer-to-peer approach to resource location in grid environments," in *Grid Resource Management*. Norwell, MA: Kluwer, 2003.
[20] I. J. Taylor, *From P2P to Web Services and Grids: Peers in a Client/Server World*. New York: Springer, 2004.
[21] A. Padmanabhan, S. Wang, S. Ghosh, and R. Briggs, "A self-organized grouping (SOG) method for efficient grid resource discovery," in *Proc. 6th IEEE/ACM GRID*, Seattle, WA, November 2005, pp. 312–317.
[22] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "The organic grid: Self-organizing computation on a peer-to-peer network," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 35, no. 3, pp. 373–384, May 2005.
[23] K. Sycara, "Multiagent systems," *Artif. Intell. Mag.*, vol. 10, no. 2, pp. 79–93, 1998.
[24] P. Grasse, "La reconstruction du nid et les coordinations inter-individuelles chez belicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs," *Insectes Sociaux*, no. 6, pp. 41–84, 1959.
[25] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A framework for the development of agent-based peer-to-peer systems," in *Proc. 22nd ICDCS*, Washington, DC, 2002, pp. 15–22.
[26] A. S. Cheema, M. Muhammad, and I. Gupta, "Peer-to-peer discovery of computational resources for grid applications," in *Proc. 6th IEEE/ACM GRID*, Seattle, WA, Nov. 2005, pp. 179–185.
[27] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," in *Proc. 14th IEEE HPDC*, Research Triangle Park, NC, Jul. 2005, pp. 113–124.
[28] A. Andrzejak and Z. Xu, "Scalable, efficient range queries for grid information services," in *Proc. 2nd P2P*, Washington, DC, 2002, pp. 33–40.
[29] S. Garnier, J. Gautrais, and G. Theraulaz, "The biological principles of swarm intelligence," *Swarm Intell.*, vol. 1, no. 1, pp. 3–31, 2007.
[30] K. Lerman, A. Galstyan, A. Martinoli, and A. Ijspeert, "A macroscopic analytical model of collaboration in distributed robotic systems," *Artif. Life*, vol. 7, no. 4, pp. 375–393, 2002.



Agostino Forestiero (A'09) received the Laurea degree and the Ph.D. degree in computer engineering from the University of Calabria, Arcavacata di Rende, Italy, in 2002 and 2007, respectively.

He has been a Researcher with the Institute of High Performance Computing and Networking of the Italian National Research Council, ICAR-CNR, Cosenza, Italy, since 2003. He coauthored over 40 scientific papers published in international journals and conference proceedings. His research interests include grid and cloud computing, P2P networks, and swarm intelligence.

Emilio Leonardi (S'09) received the Laurea degree in electronics engineering and the Ph.D. degree in telecommunications engineering from the Politecnico di Torino, Torino, Italy, in 1991 and 1995, respectively.

He is currently an Associate Professor with the Department of Electronics, Politecnico di Torino. In 1995, he was with the Department of Computer Science, University of California, Los Angeles. In the summer of 1999, he was with the High Speed Networks Research Group, Lucent Technology-Bell Labs, Holmdel, NJ, and in the summer of 2001, he was with the Department of Electrical Engineering, Stanford University, Stanford, CA. He coauthored over 100 papers published in international journals and presented in leading international conferences. His areas of interest are all-optical networks, queueing theory, and scheduling policies for high-speed switches.

Carlo Mastroianni (M'06) received the Laurea and the Ph.D. degrees in computer engineering from the University of Calabria, Arcavacata di Rende, Italy, in 1995 and 1999, respectively.

He has been a Researcher with the Institute of High Performance Computing and Networking of the Italian National Research Council, ICAR-CNR, Cosenza, Italy, since 2002. Previously, he was with the Computer Department of the Prime Minister Office, Rome, Italy. He coauthored over 80 papers published in international journals and conference proceedings. He edited special issues for the journals *Future Generation Computer Systems*, *Journal of Network and Computer Applications*, and *Multiagent and Grid Systems*. He is the ICAR-CNR coordinator for the European CoreGRID Network of Excellence and the related ERCIM Working Group. His areas of interest are grid and cloud computing, P2P networks, bio-inspired algorithms, and multiagent systems.

Michela Meo (M'03) received the Laurea degree in electronics engineering and the Ph.D. degree in electronic and telecommunications engineering from the Politecnico di Torino, Torino, Italy, in 1993 and 1997, respectively.

Since November 1999, she has been an Assistant Professor with the Politecnico di Torino. She coauthored more than 120 papers, about 40 of which are in international journals. She edited six special issues of international journals, including *Mobile Networks and Applications*, *Performance Evaluation*, and *Computer Networks*. Her research interests are in the fields of performance evaluation and modeling, traffic classification and characterization, P2P, and green networking.

Dr. Meo was Program Co-Chair of two editions of ACM MSWiM, General Chair of another edition of ACM MSWiM, and Program Co-Chair of IEEE QoS-IP, IEEE MoVeNet 2007, and IEEE ISCC 2009. She was also in the program committee of about 50 international conferences, including Sigmetrics, IEEE INFOCOM, ICC, and Globecom.