Self-Chord: a Bio-Inspired Algorithm for Structured P2P Systems

Agostino Forestiero, Carlo Mastroianni
ICAR-CNR, Rende(CS), Italy
{forestiero,mastroianni}@icar.cnr.it

Michela Meo
Politecnico di Torino, Italy
michela.meo@polito.it

Abstract

This paper presents "Self-Chord", a bio-inspired P2P algorithm that can be profitably adopted to build the information service of distributed systems, in particular of Computational Grids. Self-Chord inherits the ability of Chord-like structured systems for the construction and maintenance of an overlay of peers, but features enhanced functionalities deriving from the activity of ant-inspired mobile agents, such as autonomy behavior, self-organization and capacity to adapt to a changing environment. Self-Chord features three main benefits with respect to classical P2P structured systems: (i) it is possible to give a semantic meaning to keys, which enables the execution of "class" queries, which are very frequent in Grids; (ii) the keys are fairly distributed over the peers, thus improving the balancing of storage responsibilities; (iii) maintenance load is reduced because, as new peers join the ring, the mobile agents will spontaneously reorganize the keys in logarithmic time. The efficiency and effectiveness of Self-Chord have been assessed both with a simulation framework and a prototype of the system.

1. Introduction

The information service is an important component of distributed computing systems, such as computational Grids [7], since it provides information and enables the discovery of the resources that can be used to build and run complex applications. The dynamic nature of Grids makes human administrative intervention difficult or even unfeasible and centralized information services are proving unfit to scale to hundreds or thousands of nodes. To tackle these issues, the scientific community has proposed to design information services according to the P2P paradigm, which offers better scalability and adaptivity features [15, 10].

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 [1]. 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 with specific hosts, often through *Distributed Hash Tables*. For example, in Chord [14], each peer is assigned a binary code, called index or key, by a hash function, and peers are organized in a ring and ordered following the values of their indexes. Resources are also indexed by keys, and each resource is consigned to the first peer on the ring whose index is equal or larger than the resource key. Structured systems are generally more efficient in terms of search time and network load but 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* [4], often inspired by biological systems such as ant colonies and insect swarms. Ant algorithms are one of the most popular examples of "swarm intelligence" systems, in which a number of ant-inspired agents follow very simple rules with no centralized control, and complex global behavior emerges from their local interactions [2]. Recently, bio-inspired techniques have been proposed to design "self-structured" P2P systems, in which the association of keys with hosts is not pre-determined but adapts to the modification of the environment [6, 9].

This paper presents *Self-Chord*, a P2P system that inherits from Chord the ability to maintain a structured ring of peers, but features enhanced functionalities achieved through the activity of ant-inspired mobile agents. 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 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, like in Chord, exploiting the pointers of the *finger tables* [14]. Therefore this basic functionality is unaltered. However, Self-Chord features several benefits with respect to Chord:

(i) In Self-Chord, peer indexes and resource keys are defined independently and 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 by a hash function. A user can issue "class" queries, i.e., explore the network to find a number of resources belonging to the same class and then select the most appropriate for his/her purpose. This is a frequent issue in Grids: for example, a user might search for hosts for which the CPU speed and the memory size are within a specified range, and successively choose among the discovered results. Moreover, the flexibility assured by Self-Chord for the assignment of keys to resources enables a semantic meaning to be given to keys. For example, each bit in the resource key may represent the presence/absence of a given *topic* [12]: this is appropriate if resources are documents, because it is possible to specify the topics on which a given document focuses.

(ii) 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 that are actually present in the system, thus fostering a fair balancing of storage responsibilities.

(iii) 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.

It should be noted that, while the presented system is inspired by Chord, similar algorithms can be defined for any structured system. For example, in CAN [13], the peers are organized in a logical multi-dimensional space, and each peer "owns" a distinct zone in this space. Each resource index is represented as a point of the space and is consigned to the peer that is responsible for the zone that contains that point. A bio-inspired algorithm can be devised to improve the flexibility of CAN: the agents would traverse the network across the n-dimensional structure to reorder the keys in a self-organizing fashion.

The rest of the paper is organized as follows: Section 2 gives an overview of the Self-Chord model; Section 3 describes the operations of ant-inspired mobile agents; Section 4 analyzes the performance of Self-Chord by presenting a set of results obtained with simulation experiments, with particular emphasis given to important features such as scalability, load balancing and dynamic behavior; Section 5 concludes the paper.

2 The Self-Chord 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 [14] 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 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 hash function. Alternatively, resource keys can be given a semantic meaning: for example, the value of each bit indicates the presence/absence of a specific topic, 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. Consequently, there is no obligation to assign a key to the peer having the same index, or to its successor, as in Chord. To inherit the efficiency of resource discovery operations offered by Chord, the resource keys must be sorted on the ring. Whereas in Chord sorting is the outcome of a global planning, in Self-Chord it is obtained through the operations of ant-inspired agents that move the resource keys across the ring.

For their work, the agents use the concept of peer *centroid*. The centroid of a peer is defined as the real value, between 0 and N_c, which minimizes the average distance between itself and all the keys stored by this peer and 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 centroid value is an indication about the keys stored in the local region of the ring and is used by

[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.

agents to move the keys. The agents tend to take a key out of a peer if its value is distant from the peer centroid, and tend to forward this key towards a peer whose centroid is as close as possible to the key value. These simple operations are performed on the basis of local information, and gradually achieve the global sorting of the keys. The details are discussed in Section 3.

Agents are generated and die like the real ants from which they are inspired. Each peer, at the time that it connects to the network, generates an agent with a given probability P_{gen}. With equal probabilities, this agent will be "right-handed" or "left-handed", meaning that it will move in clockwise or counterclockwise direction. The lifetime of the agent is randomly generated with a statistical distribution whose average is equal 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, the average number of agents $\overline{N}_a$ that circulate in the network at a given instant of time is associated with the average number of peers present in the network at the same time, $\overline{N}_p$,

$$\overline{N}_a \cong \overline{N}_p \cdot P_{gen} \tag{1}$$

The keys are sorted on the ring, and the obtained order is robust with respect to successive modifications of the environment, for example to the connections/disconnections of peers. The sorting of keys allows Self-Chord to rapidly serve discovery requests, because a search message can be driven towards the peer that stores the desired keys. Both the sorting process and the discovery procedures exploit Chord-like *finger tables*, so as to assure logarithmic times, as Sections 3 and 4 will show.

Figure 1 gives an example of the way resource keys are sorted. In this sample scenario, the values of B_p and B_c are respectively equal to 6 and 3, and the ring contains 16 peers. At the interior of the ring, the figure specifies the indexes of the peers, whereas at the exterior it reports, for every 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 both the values of centroids and peer indexes are sorted in clockwise direction, but they are not related to one another. Indeed, different approaches are used to sort them: the peer indexes are sorted by the Chord management operations, whereas the resource keys are sorted by the self-organizing operations of the Self-Chord agents.

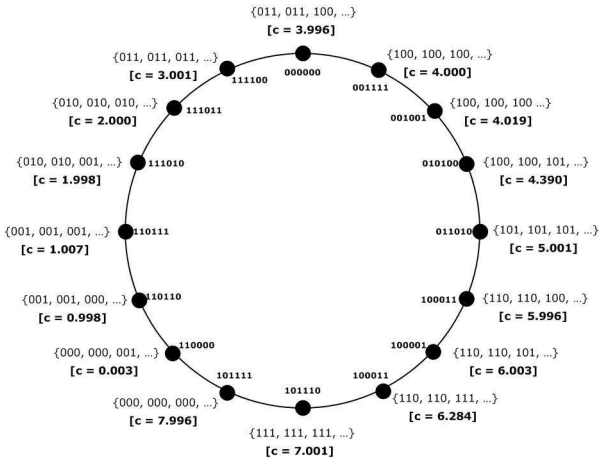

Figure 1. Sample sorting of resource keys in the peers of Self-Chord. For each peer, its index, a number of stored keys and the centroid are reported.

3 Operations of Self-Chord Agents

Each agent, in its lifetime, performs a few simple operations, cyclically: (i) while it is not carrying any key, it hops from a peer to its predecessor or successor, depending on the agent being left-handed or right-handed; (ii) at any new peer, it decides whether or not to *take* a key out of the peer; (iii) after taking a key, the agent jumps to a new peer exploiting the current peer's finger table; (iv) at the new peer, the agent decides whether or not to *leave* the carried key. Operations (ii) and (iv) are repeated until the agent takes or leave a key, respectively.

The decision about the take operation depends on the values of the key under consideration and the centroid of the current peer. To foster the sorting of keys, it is convenient to take keys that are distant from the peer centroid, whereas the keys that are close to it are probably already placed in the correct place. Therefore, the probability of taking a key r out of a peer having centroid c is defined to be inversely proportional to the similarity between r and c. The similarity function $f(r, c)$ and the *take* probability P_{take} are,

$$f(r, c) = 1 - \frac{d(r, c)}{N_c/2} \tag{2}$$

$$P_{take} = \frac{k_t}{k_t + f(r, c)} \quad \text{with } 0 \leq k_t \leq 1 \tag{3}$$

where $d(r, c)$ is the distance between r and c, computed on the circular space of the keys. For example, with N_c=64, $d(12, 18.7)$=6.7 and $d(3, 63.5)$=3.5. The value of $f(r, c)$ is comprised between 0 (maximum diversity between r and c) and 1 (maximum similarity). With high probability the agent takes a key whose value is distant from the peer centroid. The parameter k_t can be tuned to modulate the take probability. In fact, the probability is equal to 0.50 when

the values of k_t and $f(r,c)$ are comparable, whereas it approaches 1 when $f(r,c)$ is much lower than k_t (i.e., when the key r is very different from the peer centroid) and 0 when $f(r,c)$ is much larger than k_t (i.e., when the key r is similar to the centroid). In this work, k_t is set to 0.1.

Once an agent has taken a key r from a peer, it tries to go to the region of the ring where this key should be deposited, in other words it tries to jump towards 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 peer indexes are ordered and the resource keys are also being ordered. First, the agent calculates the difference $r - c$ in the arithmetic modulo N_c, where 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]:

$$\frac{r-c}{N_c} = \frac{P_d - P_s}{N_r} \tag{4}$$

Accordingly, the agent tries to jump to a peer whose index is as close as possible to:

$$P_d = P_s + \frac{N_r}{N_c}(r-c) \tag{5}$$

To do this, the agent exploits the *finger table* of P_s. In Chord, the $i-$th finger of peer p, denoted by $p.finger(i)$ contains the index of the first peer, d, that succeeds the index of p by at least 2^{i-1}, namely $d = successor(p+2^{i-1}), i = 1..B_p$. The finger table is used by Chord to let the search messages jump to distant peers, so as to complete discovery procedures in a logarithmic time, since at every jump the search space can be halved. Self-Chord uses a *bidirectional* finger table, in which a *reverse* finger table is defined to point to the peers that follow the current peer in the *counterclockwise* direction. A reverse finger, denoted as $p.rev\_finger(i)$, points to the peer with index $\overline{d} = predecessor(p-2^{i-1}), i = 1..B_p$. The reverse fingers are symmetrical to those used by Chord and can be easily maintained with the only additional cost of doubling the storage memory for the fingers. A similar structure was defined in the *BiChord* system [8].

After calculating P_d, the agent jumps to the peer of the finger table whose index is the closest to P_d. At the new peer, the agent evaluates the *leave* operation (see the details below). If this operation is actually performed, the agent will again move towards the successor or predecessor peer, until it will take another key. Otherwise, the agent will recalculate the value of P_d and make another jump, trying to approach better the region of the ring where the carried key should be deposited.

[2] In formula (4), N_r is the number of potential index values that can be assigned to a peer, and is equal to 2^{B_p}

The reason why a reverse finger table is used is now explained. While in Chord it is always possible to choose a finger that points to a peer whose index is not higher than the target peer (the target peer being the peer that stores the desired keys), this cannot be assured in Self-Chord, as the placement of keys over the ring is based on the agents' statistical operations, not on 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 clockwise direction to return to the target peer. With a bidirectional finger table, an agent can move in both directions, so this problem does not occur.

After each jump, the agent must decide whether or not to leave the key on this peer. The *leave* probability, P_{leave}, is,

$$P_{leave} = \frac{f(r,c)}{k_l + f(r,c)} \quad \text{with } 0 \leq k_l \leq 1 \tag{6}$$

where r is the value of the carried key, c is the centroid of the current peer, and the similarity function $f(r,c)$ is computed as in (2). Contrary to P_{take}, P_{leave} 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. The parameter k_l is set to a higher value than k_t, specifically to 0.5, in order to limit the frequency of leave operations. Indeed, it was observed that if the leave probability function tends to be too high, it can be difficult for an agent to carry a key for an amount of time sufficient to move it into the appropriate Grid region.

Take and leave operations contribute to the correct reordering of keys, because the agents tend to place every key in a peer that has a centroid value close to the key value. The progressive sorting is 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.

4 Performance Analysis of Self-Chord

To assess the Self-Chord algorithm, a set of experiments were performed with an event-based simulator that has already been used for other bio-inspired algorithms [6]. Simulation results have also been validated, for small networks, against those obtained with a prototype of Self-Chord that is available at the Web site *http://self-chord.icar.cnr.it*.

An efficient method to evaluate the Self-Chord sorting process is to consider the distances (in the space of resource keys) between the centroids of every two consecutive peers, and compute the average of these values. In fact, when the keys belonging to N_c classes are correctly sorted across a ring of N_p peers, the centroid values of the peers should be sorted and equally spaced, and the distance between any two consecutive centroids should be comparable to N_c/N_p.

A first set of tests were performed in networks having a varying number of peers, from 256 to 4096. It is assumed that B_c=10 (resources are categorized into N_c=1024 classes) and B_p=16 (peer indexes are defined over 16 bits). It is also assumed that the average number of resources published by a peer, referred to as $\overline{N_{res}}$, is equal to 10 and the actual number of resources of each single peer is extracted with a Gamma probability function. The key value of each published resource is generated with a uniform distribution, therefore at the beginning key values are distributed randomly; afterwards, the keys are sorted through the operations of Self-Chord agents. The P_{gen} probability is set to 1.0: each new or reconnecting peer issues exactly one agent. The dynamic characteristics are modeled as follows: each peer has a different average connection time, and the global average for all the peers, T_{peer}, is set to 5 hours. 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 the next peer 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. It should be noted here that all these settings, as well as the setting of the parameters k_t and k_l, discussed in Section 3, can affect the duration of the transient phase, but do not influence the behavior of Self-Chord in the steady state.

Figure 2 shows the trend of the average distance between consecutive centroids. The figure shows that, starting at time 0 from a state with maximum disorder, and owing to agent operations, the mean of the centroid distance decreases from very large values to the expected value N_c/N_p, confirming the capacity of the Self-Chord algorithm to order the keys on the ring. Of course, the time needed to reorder the keys increases with the number of peers. It ranges from less than 2000 time units with 256 peers to about 12,500 time units with 4096 peers. To obtain the value in seconds, the number of time units must be multiplied by the agent forwarding time T_{mov}. These results show 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, and a new key can be moved to the correct peer in logarithmic time. This issue will be better analyzed in Section 4.2.

The sorting of keys over the ring is profitably exploited by the discovery procedure. A search message is issued by a peer to find as many keys as possible that have a specified value, and therefore belong to a given class. This is obtained by driving the search message towards the peer whose centroid is the closest to the target key value. Indeed, at the steady state, the values of the keys stored by a peer are always very close to the peer centroid (the analysis of this phenomenon can be found in [5]). At each step, the search message is forwarded, through the finger tables, to the peer whose centroid is estimated to be the closest to the target key value. The destination peer is selected through a proportion between the resource keys and the peer indexes, similarly to what is done in the reordering phase, see (4) and (5). If the centroid of the destination peer is closer to the target key than the centroid of the current peer, the search message is forwarded to that peer, and the discovery procedure continues. Whenever this condition is not satisfied, the discovery procedure terminates, because with very high probability the current peer is the one that stores the largest number of keys having the desired value.

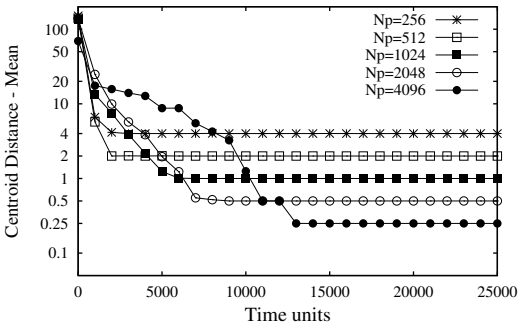

Figure 2. Average distance between two consecutive centroids with N_c=1024 resource classes and variable number of peers. The average distance tends to the value N_c/N_p.

In an ordered ring, the number of steps that are 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 [14]. Figure 3 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 $\frac{1}{2}\lg_2 N_p$ [14], but it is reduced to $\frac{1}{3}\lg_2 N_p$ in BiChord [8], in consequence of the presence of the reverse finger table. Figure 3 shows that also in Self-Chord the average number of steps is always very close or slightly larger than $\frac{1}{3}\lg_2 N_p$. Moreover, the 99th percentile is always lower than $\lg_2 N_p$, meaning that the search process is very fast also in the most unfortunate cases.

Figure 4 shows the mean number of keys discovered by a search request, for different values of N_p. The assumption

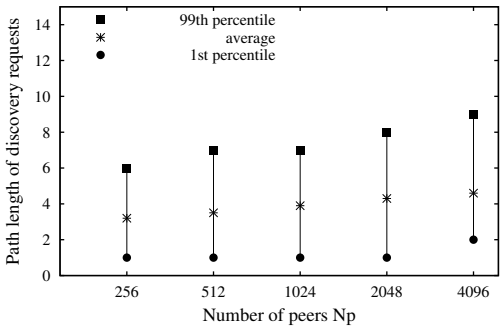

Figure 3. Path length of search messages with variable number of peers. The average, the 1st and the 99th percentile are reported.

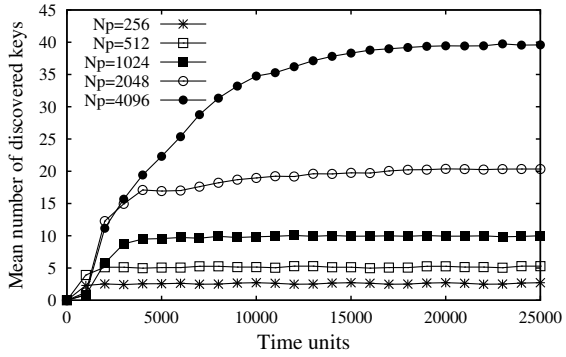

Figure 4. Average number of keys discovered by a query vs. time, with variable number of peers.

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, due to the statistical nature of the reordering process, it is possible that these neighbors store a low number of keys having the desired value. In Figure 4, the number of discovered keys is reported versus time, starting from a situation of maximum disorder, in order to show that the index gradually increases as reordering process proceeds. The steady value is comparable to $(N_p \cdot \overline{N}_{res})/N_c$. In fact, this is the expected number of keys of a specific class that are published in a network having N_p peers, in the case that $\overline{N}_{res}$ 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 almost all the resources that have the desired value of the key.

4.1 Non-Uniform 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 uniformly distributed. However, 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.

A set of experiments was performed assuming that the key values are distributed according to the triangular distribution shown in Figure 5.

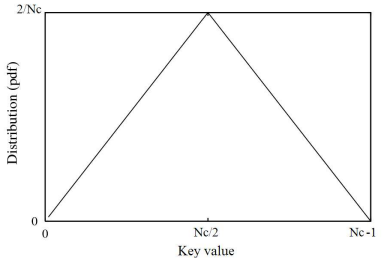

Figure 5. Triangular distribution of keys.

In classical structured P2P systems, a non-uniform distribution of keys produces a non-uniform balance of load. In Chord, for example, under the described triangular distribution, the peer with index $\frac{N_c}{2}$ 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, both with uniform and non-uniform distribution of keys. In fact, the agents take and leave the keys with no regard to their relative popularity.

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 [14], is about 50, compared to the value of 22 experienced in Self-Chord. With a non uniform 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. Conversely, Self-Chord does not need any superstructure to achieve a fair load balance.

Previously it was mentioned that the resource discovery algorithm estimates the index of the next peer to which a search message is forwarded, with the implicit assumption of a uniform distribution of keys. The discovery procedure could become longer with a non-uniform distribution, be-

cause the destination peer could have a different centroid value than the estimated one. Therefore, a set of experiments was performed to observe what happens if the distribution of keys is triangular. Figure 6 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 a possible erroneous estimation of the centroid value of the destination peer can be rapidly compensated by the next steps of the search message. Indeed, the average number of steps required with the non-uniform case is only slightly larger than that obtained with the uniform distribution, and in the most unfortunate cases (evaluated through the 99th percentile) a few more steps are sufficient to successfully complete the discovery procedure. More details about the behavior of Self-Chord in the case of non-uniform distribution can be found in [5].

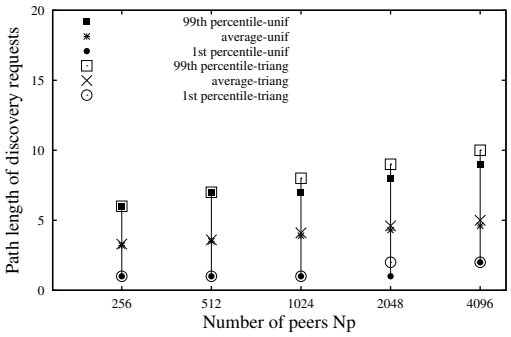

Figure 6. Path length of discovery requests: average, 1st and 99th percentile calculated with uniform and triangular distributions of keys and a variable value of N_p.

4.2 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: the keys of the new resources will be picked by the agents that pass by this peer. The processing load L can be defined as the average number of agents per second that arrive and are processed at a peer. L can be calculated by multiplying the average number of agents $\overline{N}_a$ by the frequency of their movements $1/T_{mov}$, 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,

$$L = \frac{\overline{N}_a}{\overline{N}_p \cdot T_{mov}} \approx \frac{P_{gen}}{T_{mov}} \quad (7)$$

The simplification is given by applying (1).

For example, if the average value of T_{mov} is equal to 5 seconds, and P_{gen} is set to 1.0, each peer receives and processes about one agent every 5 seconds, which is an acceptable load, since take and leave operations are very simple. This result, obtained theoretically, has been confirmed by simulation. Note that the processing load only depends on the probability that a reconnecting peer generates an agent, P_{gen}, and on the frequency of agent movements across the Grid, $1/T_{mov}$. It does not depend on other system parameters such as the frequency of peer joinings and disconnections, the network size, the average number of resources published by a node and so on, which confirms the scalability properties of Self-Chord.

The results discussed so far have shown that the Self-Chord agents 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 few agent operations that move the new keys to the correct place of the ring. The relocation of a new key is achieved by a procedure that is analogous to that used for resource discovery, and therefore can be completed very rapidly, in a time that is less than logarithmic with respect to the number of peers. However, a set of specific experiments was performed to evaluate 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. Figure 7 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 number of 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

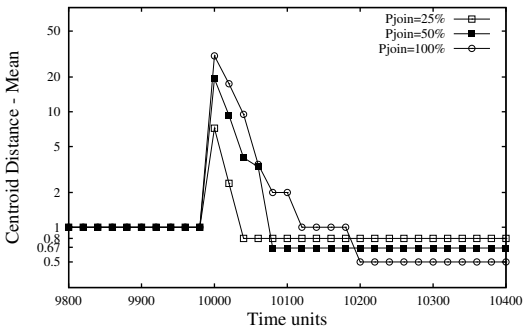

Figure 7. Average distance between two consecutive centroids. At the beginning, the values of N_p and N_c are set to 1024. After 10,000 time units, a percentage P_{join} of new peers join the network.

new value of N_c/N_p. With N_c 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 Figures 7 and 2 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.

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.

4.3 Discussion

The results reported in this section allow for a comprehensive analysis of Self-Chord and its comparison with Chord. Due to the self-organizing sorting of keys performed by agents, the most important functionality of Chord, logarithmic discovery time, is preserved. In addition, Self-Chord features several benefits with respect to Chord:

(i) *Better support of complex discovery requests*. In structured P2P systems, Chord included, a user can search for a specific resource using its resource key as a search parameter, but more complex discovery requests, such as range queries, are not easily supported, unless at the cost of maintaining complex structures, for example tree-shaped [11]. In Self-Chord, the definition of resource keys is flexible, and it is possible to give them a semantic meaning. This enables the system to serve both "class" queries, issued to search for resources having common characteristics, and "range" queries, issued to discover resources that may overlap contiguous classes. Both these functionalities are very useful in Grid Computing. Range queries can be supported if key values of resources are given a semantic meaning, or if they are obtained by a *locality preserving* hash function, as for example in [3]. With both approaches, keys of similar resources are placed into neighbor hosts, and in many cases may be discovered by a single query.

(ii) *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 does not depend neither on the distance from its predecessor nor on the popularity distribution of keys. As discussed in Section 4.1, 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 non-uniform distribution.

(iii) *Improved dynamic behavior*. In Chord the computational load strongly depends on the dynamic behavior of the system, for example on the churn rate of peers, whereas in Self-Chord it is constant. In Self-Chord, the placement of new/modified keys in the correct position of the ring is achieved in a logarithmic time, so it is as fast as a resource discovery operation. Moreover, any perturbation of the steady condition, even very intense, such as those considered in Section 4.2, is efficiently managed, and the key ordering is recovered rapidly. This assures scalability (keys are continuously reordered as the network grows) and robustness with respect to environmental changes.

Finally, 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.

5 Conclusions

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. The efficiency and effectiveness of Self-Chord

are confirmed by results obtained by simulation. In this paper the presented ant-inspired approach is applied to 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 multi-dimensional grids or trees. In these cases, the self-organization and ordering of keys can be achieved with proper modifications of the bio-inspired algorithm presented in this paper.

References

[1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

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

[3] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: A multi-attribute addressable network for grid information services. In *Proc. of the Fourth International Workshop on Grid Computing GRID '03*, 2003.

[4] D. C. Erdil, M. J. Lewis, and N. Abu-Ghazaleh. An adaptive approach to information dissemination in self-organizing grids. In *Proc. of the International Conference on Autonomic and Autonomous Systems ICAS'06*, 2005.

[5] A. Forestiero, C. Mastroianni, and M. Meo. Self-Chord: Self-Structured Management of Resources on a Structured Ring of Peers. RT-ICAR-CS-08-08, CNR Institute of High Performance Computing and Networking. *http://biblio.cs.icar.cnr.it/biblio/tr/scaricaTR.asp?FileID=78*, November 2008.

[6] A. Forestiero, C. Mastroianni, and G. Spezzano. So-grid: A self-organizing grid featuring bio-inspired algorithms. *ACM Transactions on Autonomous and Adaptive Systems*, 3(2), May 2008.

[7] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[8] J. Jiang, R. Pan, C. Liang, and W. Wang. Bichord: An improved approach for lookup routing in chord. In *Proceedings of the 9th Conference on Advances in Databases and Information Systems, ADBIS 2005*, volume 3631 of *Lecture Notes in Computer Science*, Tallinn, Estonia, 2005. Springer.

[9] S. Y. Ko, I. Gupta, and Y. Jo. A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing. *ACM Transactions on Autonomous and Adaptive Systems*, 3(3):1–34, 2008.

[10] C. Mastroianni, D. Talia, and O. Verta. Designing an information system for grids: Comparing hierarchical, decentralized p2p and super-peer models. *Parallel Computing*, 34(10):593–611, October 2008.

[11] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing HPDC 2005*, Research Triangle Park, NC, USA, July 2005.

[12] C. Platzer and S. Dustdar. A vector space search engine forweb services. In *Proc. the Third European Conference on Web Services ECOWS '05*, 2005.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[14] 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. of the Conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM'01*, 2001.

[15] I. J. Taylor. *From P2P to Web Services and Grids: Peers in a Client/Server World*. Springer, 2004.