Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Self-Chord: Self-Structured Management of Resources on a Structured Ring of Peers

Agostino Forestiero[1], Carlo Mastroianni[1], Michela Meo[2]

RT-ICAR-CS-08-08 **Novembre 2008**

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS)
[2] Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Self-Chord: Self-Structured Management of Resources on a Structured Ring of Peers

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 organize 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. The agents, through simple local operations driven by probabilistic choices move resource keys across the ring of peers, and sort them in a self-organizing fashion. 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" and range queries; (ii) the keys are fairly distributed over the peers, thus improving the balancing of storage responsibilities; (iii) maintenance load is reduced because it is not necessary to immediately place the keys published by new or reconnecting peers: the mobile agents will spontaneously reorganize the keys in a time that is logarithmic with respect to the network size. The efficiency and effectiveness of Self-Chord have been assessed with a simulation framework and a prototype, which is available at the Web page http://self-chord.icar.cnr.it.

1 Introduction

The information service is an important component of distributed computing systems, such as computational Grids [11], 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 [23].

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 [2]. 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 [21], 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, to the first following peer, 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* [9], 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. [5]. Recently, these techniques have

been proposed to design "self-structured" P2P systems, so called because the association of keys with hosts is not predetermined but adapts to the modification of the environment [10, 16].

This paper presents *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. Self-Chord does not place resource keys to specified hosts, as Chord does: this feature is actually unnecessary and strongly 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. They act on the basis of probabilistic choices, which are driven by the system state in the local region. The sorting of keys allows discovery operations to be executed in logarithmic time, exploiting the pointers (the *finger tables*) provided by the Chord structure. In Self-Chord, the length of the next jump of a query message is calculated by making a proportion between the peer indexes, which are ordered over the ring, and the resource keys, which are also ordered in an independent fashion. Simulation results show that the number of hops of discovery messages is comparable to that experienced in Chord, therefore this basic functionality is unaltered. However, the bio-inspired approach of Self-Chord leads to several benefits with respect to Chord [21], as illustrated in the following:

(i) In Chord, both peer and resource keys are mapped by hash functions into the same number of bits, which is usually very large. Each resource is given a unique key, and is assigned to a well specified host. This obliges users to search for a specific resource, using its resource key as a search parameter, but it is not possible to issue a query for a class of resources. The same restriction is also present in other structured P2P systems. In Self-Chord, there is no relation among the space of resource keys and the space of peer indexes, because they can be defined on different numbers of bits. In this way it becomes possible to define "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. This way, a user can 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 Grid 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.

(ii) The flexibility assured by Self-Chord for the definition of the space of resource keys enables a semantic meaning to be given to them. For example, each bit in the resource key may represent the presence/absence of a given *topic* [19]: this is appropriate if resources are documents, because it is possible to specify the topics on which a given document focuses. Alternatively, a resource or service can be mapped into a binary key by a *locality preserving* hash function, as for example in [6]. If such functions are used in Self-Chord, resources that are similar, but not enough to belong to the same class, are assigned similar keys. Both methods allow "range queries" to be executed efficiently: since keys related to similar resources are placed into neighbor hosts, they can easily be found with a single discovery request.

(iii) Structured systems can produce imbalance problems depending on the location of peers and the distribution of keys. As already mentioned, in Chord a resource is managed by the peer whose index is equal to the resource key or by its *successor* on the ring. Therefore, a peer might store a large number of keys if the distance between this peer and its predecessor is large. If keys are not uniformly distributed, because 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. Instead, the keys are fairly distributed over the peers that are actually present in the system, thus fostering a fair balancing of storage responsibilities. Moreover, results show that the efficiency of discovery operations is not affected by a non-uniform distribution of keys.

(iv) 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 also after the correct reordering has been achieved, and will spontaneously reorganize the keys. Therefore, in Self-Chord the computational load is constant, whereas in Chord it strongly depends on the dynamic behavior of the system, for example on the churn rate of peers. 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. Experiments have been performed to assess the system reaction in the case that a large number of peers join the ring at the same time. It was found that the keys are reordered very rapidly, which means that Self-Chord is scalable (keys are continuously reordered as the network grows) and robust with respect to environmental changes.

All the mentioned results were obtained both with an ad hoc event-based simulator and with a prototype available at the Web site http://self-chord.icar.cnr.it. This prototype is based on the Open-Chord open source implementation of the Chord system, available at http://open-

chord.sourceforge.net/.

This paper aims to make a novel contribution with regard to two aspects. First, it opens a new research avenue for P2P frameworks, because it presents a P2P system that inherits the beneficial characteristics of structured systems, but exploits a self-organizing algorithm to distribute the resource keys on the structure, which improves scalability, flexibility and load balancing characteristics. It should be noted that, while the presented system is partly based on Chord, similar algorithms can be defined for any structured system. For example, in CAN [20], resources are placed in a multi-dimensional structure: the position of a resource on each dimension is computed from the value of a corresponding numerical parameter. A bio-inspired algorithm can be devised also in this case: the agents would traverse the network across the n-dimensional structure to reorder the keys.

Secondly, Self-Chord fosters the integration of Grid and P2P worlds. In fact, it supports class queries, issued to discover the resources that belong to a given class, as well as range queries. These queries are typical in Grid systems. When a Grid complex application has to be designed, a user generally needs to collect a number of basic services that will be orchestrated to compose the application. The ultimate selection must be made at run time: owing to the volatility and dynamic nature of Grid resources and services, it is possible that some of them become unavailable or their characteristics change with time: in this case, alternative resources, among those collected in the search phase, may be used.

The rest of the paper is organized as follows: the following section illustrates the state of the art in the mentioned fields. Section 3 gives an overview of the Self-Chord model and describes how the model organizes the resource keys on the network. Section 4 gives more details about the Self-Chord algorithm and the operations that are performed by ant-inspired mobile agents to correctly place the keys. Section 5 shows the results of simulation experiments that confirm the effectiveness of Self-Chord, with particular emphasis given to important features such as scalability, load balancing and dynamic behavior. Finally, Section 6 concludes the paper.

2 Related Work

In most distributed systems, and particularly in Grid systems, information services are implemented in accordance with centralized or hierarchical approaches, mostly because the client/server approach is still used today in the majority of distributed systems and in Web service-based frameworks. However, these approaches are impractical in large multi-institutional Grids because they undergo severe drawbacks such as poor scalability, limited autonomy of Grid 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 approaches. Novel approaches for the construction of scalable and efficient Grid information systems need to have the following properties [14, 24]: self-organization (meaning that Grid 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 [9]. In the architecture proposed in [1], 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 which are likely to be maintained by the group. This is an interesting approach but it still has non-scalable characteristics: for example, it is required that each Grid node has a link to all the leader nodes, which is problematic in a very large Grid. A self-organizing mechanism is also exploited in [7] 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 [22]. For example, insects and birds can be imitated by mobile agents that travel through the hosts of a Grid 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. It is usually achieved through a direct exchange of messages among agents, as in the *UWAgents* system [12]. Conversely, Self-Chord exploits the *stigmergy* paradigm [13]: 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 Grid, 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 some species of ants [5]. 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 [4] is tailored to the design, implementation and evaluation of P2P applications based on multi-agent 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 Grid to satisfy user requests. However, whereas in Anthill, ants are generated after user requests, in Self-Chord agents operate continuously and autonomously, since the reorganization of keys is essential for the efficient performance of discovery requests.

Self-Chord puts itself along the research avenue of P2P resource discovery algorithms. Structured P2P algorithms are usually efficient in file sharing P2P networks, but structure management can be cumbersome and poorly scalable in large and dynamic Grids, especially when the churn rate, the frequency of peer disconnections, is high. Owing to its delf-organizing nature, Self-Chord proves to be both scalable and robust with respect to environmental modifications. Moreover, one of the main objectives of Self-Chord is to serve *class* and *range* queries. The efficient management of these queries is a fundamental requirement in Grid systems, as users do not often need to discover well-defined resources, but resources having more loosely specified characteristics. For example, a query might be issued to discover a supercomputer with CPU speed comprised in a given range, or a Web service whose estimated service time is within a given interval. A naive way to manage these queries is to issue as many simple queries as are sufficient to cover all the possible values of target keys, and then collect the results. This solution is clearly inefficient, since it does not exploit the similarity of the target keys specified in the different simple queries. The efficient execution of range queries is indeed a very tough issue for Grids and P2P systems [8]. Some types of structured systems are capable of serving range queries, but often at the cost of either maintaining complex tree-like structures [18] or increasing the traffic load by issuing a number of sub-queries [3]. The Self-Chord information system naturally supports class and range queries. As mentioned in the introductory section, this feature derives from the flexibility provided by Self-Chord in the definition of the number of bits of resources keys, and from the utilization of two separate algorithms that are used to reorder the peers and the keys in an independent fashion.

3 The Self-Chord Model

In Self-Chord, peers are organized in a logical ring. Each peer is given an index that is obtained with a uniform hash function and can have values between 0 and $2^{B_p}-1$, where B_p is the number of bits in peer indexes. The ring is constructed and maintained as in Chord (see [21] for the details).

Each published resource is associated with a binary key that will be used to discover and access the resource. The values of resource keys range from 0 to $2^{B_r}-1$ and can be obtained in two ways. The first way is through the use of a *locality preserving* hash function, so that similar keys are associated with similar resources. The number of possible values of the resource key, $N_c = 2^{B_r}$, can be viewed as the number of classes in which the resources are categorized. As mentioned in the introductory section, 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. Alternatively, resource keys have a semantic meaning: for example, the value of each bit indicates the presence/absence of a specific topic, for example if the resource is a document. Note that in both cases, similar resources are given similar key values, which enables the support of "range" queries.

In Chord, B_p and B_r 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_r 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 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 agents to move the keys. In fact, 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 4.

Agents do not operate forever, but 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}. The lifetime of this

[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.

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 N_a that circulate in the network at a given instant of time is associated with the number of peers present in the network at the same time, N_p, in the following way:

$$N_a \cong N_p \cdot P_{gen} \cdot \frac{T_{peer}}{T_{agent}} \quad (1)$$

Notice that this statistical relationship does not imply that any peer is aware of the values of T_{peer} and N_p. However, the formula can be useful to tune the velocity 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, it results that $T_{agent} \cong T_{peer}$, therefore the average number of agents is P_{gen} times the number of peers connected in the ring.

After a transient phase, the keys will be sorted on the ring, as the rest of the paper will show. Moreover, the obtained order is very robust with respect to successive modifications of the environment, for example to the connections and disconnections of peers. The sorting of keys allows Self-Chord to rapidly serve discovery requests, because it is possible to send a search message towards the peer that stores the desired key. Both the sorting process and the discovery procedures exploit Chord-like *finger tables*, so as to assure logarithmic search times, as Sections 4 and 5 will show.

As mentioned before, the sorting of resource keys is not tied to the values of peer indexes. To clarify this point, Figure 1 gives an example of the way resource keys are sorted. In this sample scenario, the values of B_p and B_r 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 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. However, there is no relation between the peer indexes and the values of the keys stored in the respective peers, because different approaches are used to sort them. In fact, 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.

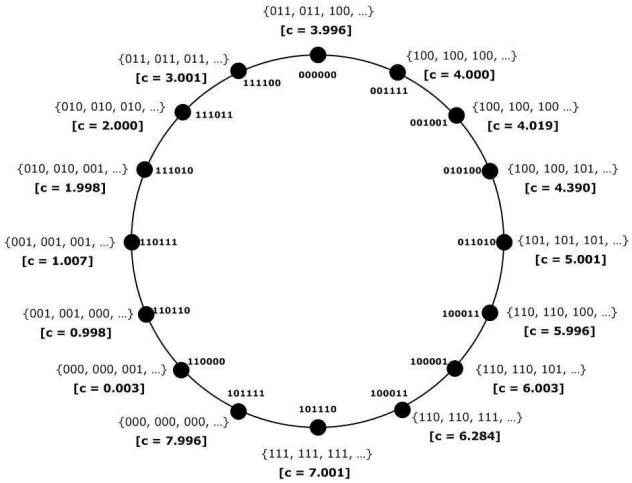

Figure 1. Distribution and ordering of resource keys in the peers of Self-Chord.

4 Operations of Self-Chord Agents

Each mobile agent gives its contribution to the reordering of resource keys over the ring. Two different approaches are discussed in the following: they will be referred to as "linear" and "logarithmic", respectively. With the linear approach, resource keys are moved by agents between adjacent peers, whereas with the logarithmic approach the agents can exploit the peer finger tables to hop towards distant peers. The logarithmic approach is understandably much faster, but it easily origins a higher imbalance in the number of keys that are stored by the peers. Therefore, it will be shown later that a hybrid strategy, which combines the two basic approaches, leads to the best performance results.

4.1 Linear Ordering of Keys

With the linear approach, 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 is not carrying 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 values r are higher that c, because these keys should be moved to *successor* peers to improve the overall ordering. These keys are identified by evaluating the condition,

$$c < r < (c + N_c/2) mod(N_c) \quad (2)$$

Conversely, a left-handed agent, which moves in counter-clockwise direction, evaluates the keys that satisfy,

$$(c - N_c/2) mod(N_c) < r < c \quad (3)$$

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

To foster the correct sorting of keys, it is convenient to pick keys that are very different from the peer centroid, whereas the keys that are similar to it are probably already placed in the correct place. Therefore, the probability of taking a key r is defined to be inversely proportional to the similarity between r and the peer centroid c. The similarity function $f(r, c)$ and the *take* probability P_{take} are,

$$f(r, c) = 1 - \frac{d(r, c)}{N_c/2} \quad (4)$$

$$P_{take} = \frac{k_t}{k_t + f(r, c)} \quad \text{with } 0 \leq k_t \leq 1 \quad (5)$$

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. This key will be carried by the agent and moved towards successor or predecessor peers, depending on the agent being right-handed or left-handed. The parameter k_t, whose value is between 0 and 1, 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 very similar to the centroid). In this work, k_t is set to 0.1, as in [5].

When an agent that is carrying a key moves to another peer, it must decide whether or not to leave the key on this peer. The *leave* probability, P_{leave}, is defined regardless the type of agent, left- or right-handed, and is,

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

where r is the value of the key carried by the agent and the similarity function $f(r, c)$ is computed as in (4). As opposed 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 is difficult for an agent to carry a key for an amount of time sufficient to move it into an appropriate Grid region. Here it is worth specifying that the values of parameters k_t and k_l have an impact on the velocity and duration of the transient phase of the sorting process, but they have little influence on the performance observed under steady conditions. In other words, Self-Chord was found to be robust with respect to the variation of these parameters.

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.

4.2 Logarithmic Ordering of Keys

With the logarithmic approach, agents move "linearly", through adjacent peers, only when they are not carrying any key. The *take* operations are performed as described in Section 4.1. However, once an agent has taken a key from a peer, it immediately tries to go to the region of the ring where this key should be deposited, in other words it tries to jump directly 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. 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]:

$$\frac{r - c}{N_c} = \frac{P_d - P_s}{N_r} \quad (7)$$

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

$$P_d = P_s + \frac{N_r}{N_c}(r - c) \quad (8)$$

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

[2] In formula (7), N_r is the number of potential index values that can be assigned to a peer, and is equal to 2^{B_r}

messages jump to distant peers, so as to speed up the discovery procedures. A search request can be served in a logarithmic time, since at every jump of the search message the search space is halved.

Self-Chord uses a *bidirectional* finger table, in which a second 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 $\overline{d} = predecessor(p - 2^{i-1}), i = 1..B_p$. This reverse finger structure is symmetrical to that used by Chord and can be easily maintained with the only additional cost of doubling the storage memory for the fingers. A similar structure has been defined in the "BiChord" system [15].

After calculating the index P_d, the agent selects the peer of the finger table whose index is the closest to P_d and uses the corresponding finger to jump to that peer. At the new peer, the agent evaluates the "leave" operation, in the same fashion as with the linear approach. If the key is deposited, the agent will again move with the linear mode, and will hop to the successor or predecessor peer, until it will take another key. Conversely, if the leave operation is not performed, the agent will make another "logarithmic" jump, trying to approach better the region of the ring where the carried key should be deposited.

The reason why a reverse finger table is used is now easily 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, 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, a key can be moved in both directions, so this problem does not occur. If the doubled storage memory in an issue, a peer can simply discard half the pointers, for example by storing only the fingers corresponding to odd values of i. The logarithmic approach of Self-Chord is hardly affected by such a reduction in the number of fingers.

Intuitively, the reordering process is performed much faster with the logarithmic approach than with the linear, as a key can be moved through longer and better directed hops. On the other hand, the linear approach allows a fairer load balance among peers to be achieved. The two approaches are compared in Section 4.3, and there it will be shown that a hybrid approach can be defined to combine the benefits of both of them.

An important consideration is that the logarithmic approach implicitly assumes that the values of resource keys are uniformly distributed. With a non-uniform distribution (some key values are more popular than others), the calculation of the target peer, in formula (8), may not be exact. However, the logarithmic nature of the process can rapidly compensate possible errors, as will be discussed in Section 5.2.

4.3 Comparison between linear and logarithmic approaches

The linear and logarithmic approaches were evaluated with an event-based simulator, for a sample scenario in which B_r=8 (resources are categorized into N_c=256 classes), B_p=12 (peer indexes are defined over 16 bits), and the number of peers actually connected in the ring is N_p=256. It is also assumed that the average number of resources published by a peer, referred to as N_{res}, is equal to 10. The actual number of resources of each single peer is extracted with a Gamma probability function.

The key value of each resource is generated with a uniform distribution, therefore at the beginning key values are distributed randomly; afterwards, the keys are sorted according to their values through the operations of Self-Chord agents. The P_{gen} probability is set to 1: it means that each new or reconnecting peer issues exactly one agent, which will be left-handed or right-handed with equal probabilities. The average time T_{mov} between two successive agent movements is set to 10 seconds: after receiving an agent, a peer forwards it to the next peer after a random interval having an average 10 seconds. The next peer can be its successor/predecessor, or another peer pointed by the finger table, depending on the adopted approach, linear or logarithmic. Moreover, it is assumed that each peer has a different average connection time, and the global average for all the peers, T_{peer}, is set to 3 hours.

To evaluate the efficacy of the Self-Chord sorting process, the distances are considered (in the space of resource keys) between the centroids of every two consecutive peers, and the mean and the standard deviation of these values are computed. In fact, when the keys are correctly sorted across the ring, the centroid values of the peers should be sorted 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 be as low as possible.

Figure 2 reports the centroid distance obtained with linear and logarithmic approaches. 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 high values to the expected value N_c/N_p, in this case equal to 1, 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 less than 25,000 seconds. On the other hand, with the linear approach, the process takes almost 300,000 seconds, a time interval more than ten times longer. Moreover, it has been found that 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.

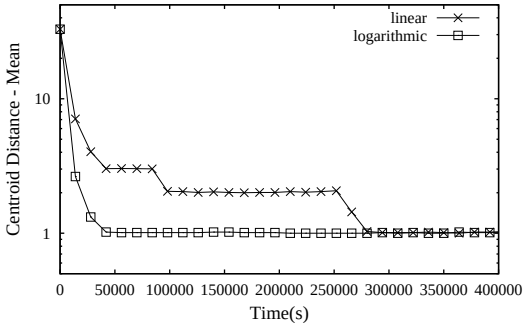

Figure 2. Average distance between two consecutive centroids with linear and logarithmic approaches.

It is also interesting to note that the reordering process experiences periods in which the average centroid distance is stable, interleaved by abrupt changes: this is more evident with the linear approach, because it is slower. 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. For example, in the case of two circles, there are two peers, typically positioned one opposite to the other in the ring, both having a centroid value equal to 0, whereas with a correct ordering there should be only one of such peers. The reordering process progressively lowers the number of circles, until it is reduced to one, 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: this explains the time intervals in which the average distance between consecutive centroid 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 repartition of load among the peers. With the linear approach, each peer receives agents exclusively from the two adjacent peers, therefore all the peers tend to store the same number of keys. Conversely, with the logarithmic approach, a peer receives the agents that carry the keys through the finger pointers of other peers. However, the number of fingers that "point" to a given peer is not a constant, but depends on the indexes of this peer and of its neighbors on the ring. In fact, peer indexes are not uniformly distributed over the ring: for any range of admissible index values, the number of peers whose indexes are comprised in this range has a logarithmic distribution [21]. As discussed in Section 4.2, a finger points to a peer index that is the successor/predecessor of an index obtained with a mathematical operation. Therefore a peer is pointed by a large number of fingers if it is the successor/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. This imbalance in the number of inbound 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 deposited with a leave operation - in the number of keys that are stored by peers.

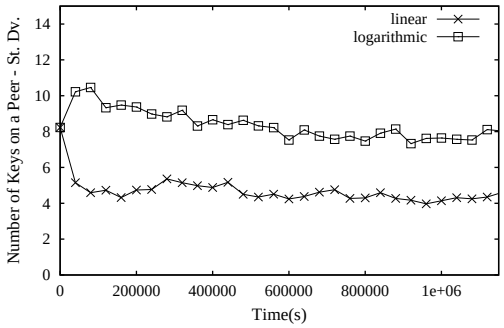

Figure 3. Standard deviation of the number of keys stored by a peer with linear and logarithmic approaches.

This is confirmed by the results in Figure 3, 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 standard deviation fluctuates around a value of 4 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.

4.4 Combined approach: switch from logarithmic to linear

The best solution would be the definition of 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 cannot be operated with a centralized approach neither it can be simultaneous for all the peers, since they cannot be aware of the global state of the system. It must be a local decision made 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, the 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 target 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 becomes lower as the curve approaches the value of N_c/N_p, as Figure 2 shows. Therefore, the derivative of the centroid distance can be used to perform the switch locally: the analysis of the derivative can be made without any knowledge on the global state of the system.

Specifically, each peer maintains a variable Δ that is updated every predetermined time interval, in our tests every 5 minutes, as follows:

$$\delta_i = \frac{C_i - C_{i-1}}{N_c} \tag{9}$$

$$\Delta_0 = D_0 \tag{10}$$

$$\Delta_i = \delta_i + F_{ev} \cdot \Delta_{i-1} \tag{11}$$

The term C_i is the centroid distance at time i, averaged in a local sector of the ring, which includes the peer itself and a small number of neighbor peers in the two directions. Therefore, δ_i is the difference between the current and the last value of the local centroid distance, normalized over the number of resource classes N_c. Since the derivative is generally negative (the average centroid distance decreases), the initial value of Δ is set to a negative value, D_0. For successive calculations, the contributions of the past values of Δ are weighed through the evaporation factor F_{ev}, whose value is between 0 and 1. The switch from the logarithmic mode to the linear mode is performed as the value of Δ exceeds a given threshold T_d that is still negative, but closer to zero than D_0. 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 largest part of the ordering process has been completed, and it is convenient to pass to the linear mode in order to better distribute the keys among the peers.

Each agent chooses its mode, "logarithmic" or "linear" according to the mode that is set on the local peer. In turn, each peer self-regulates its mode: at the beginning it sets it to "logarithmic", than switches it to "linear" as soon as the value of Δ, calculated locally, exceeds the threshold T_d. As the ordering process proceeds, the peers will gradually switch from the logarithmic to the linear mode, and so will the agents.

Figures 4 and 5 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. For this test, the parameters for the calculus of the derivative were set as follows: $D_0 = -0.1$, $F_{ev} = 0.9$ and $T_d = -0.01$. It is noticed that the combined approach accomplishes its purpose, since it reorders the keys as fast as the logarithmic approach and, in the steady situation, balances the load as well as the linear approach. The local centroid distance was calculated over a local sector of 7 peers: the current peer plus 3 adjacent peer on both sides. It is worth noting that a different setting of these parameters can cause an anticipation or delay of the switch operations performed by the peers. This can reduce or extend the transient phase, but has hardly any effect on the behavior of the system in the steady situation.

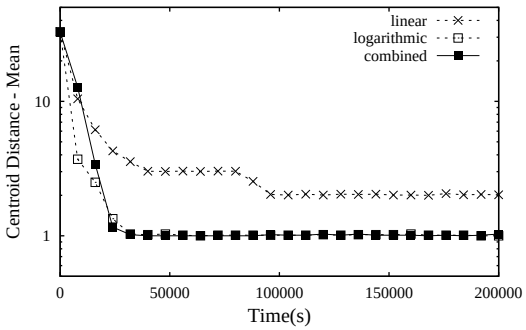

Figure 4. Average distance between two consecutive centroids with linear, logarithmic and combined approaches.

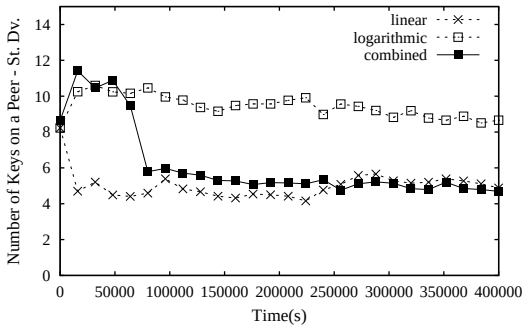

Figure 5. Standard deviation of the number of keys stored by a peer with linear, logarithmic and combined approaches.

To better illustrate the switch process, Figure 6 shows the

trend of the Δ parameter, averaged over all the peers, and in parallel the number of peers that have switched to the linear mode. Note that the largest part of peers operate their switch as the average value of Δ is around the threshold value, -0.01.

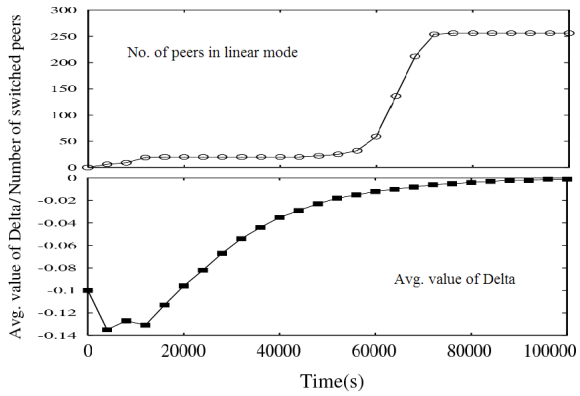

Figure 6. Analysis of the combined approach. Parallel trend of the number of peers that have switched to the linear mode and of the value of the Δ parameter, averaged on all the peers.

An important issue to tackle is the management of new resources, for example those published by the peers that join the network. To speed up the correct placement of the corresponding keys, the agents adopt the logarithmic mode when they pick the keys of new resources. In a stable situation, in which most of the keys have already been ordered, the number of steps needed to correctly place a new key is $\Theta(log(N_p))$. Though the exact value of N_p is not known by the peers, it can be overestimated to assure that a new key is moved with the logarithmic mode for at least a number of jumps equal to $log(N_p)$. After these logarithmic jumps, the agents will again base their decisions on the value of the Δ parameter maintained by each single peer.

5 Performance Analysis of Self-Chord

So far, all the tests were performed with a limited number of peers (256) because the linear approach is very slow in larger networks. In this section, however, the combined approach is adopted for all the tests, with the mode switch mechanism described in Section 4.4. This allows results to be obtained with larger and more realistic networks.

In the tests, the parameters P_{gen}, N_{res}, T_{mov} and T_{peer} and are set as in Section 4.3: their respective vales are 1.0, 10 resources per peer, 10 seconds and 3 hours. 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 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. As discussed for other parameters, 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 this phase.

The rest of this section will analyze three important aspects of Self-Chord: its scalability, its behavior with a non-uniform distribution of keys, and its dynamic characteristics.

5.1 Scalability Analysis

This section presents simulation results obtained with a variable number of peers N_p ranging from 256 to 4096. Figure 7 shows the trend of the average distance between consecutive centroids. In all the tested cases, 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.

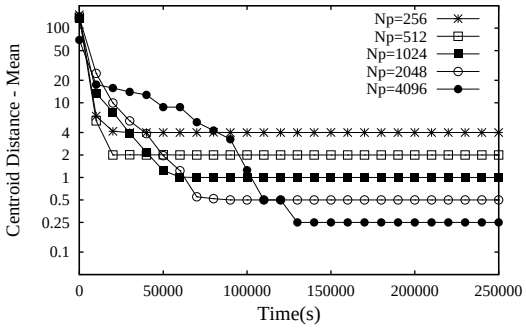

Figure 7. Average distance between two consecutive centroids with 1024 resource classes and variable number of peers.

Of course, the time needed to reorder the keys increases with the number of peers. It ranges from less than 20,000 seconds with 256 peers to about 125,000 seconds with 4096 peers. The last value, corresponding to about 35 hours, could seem long at a first sight, but it must be considered that the overall reordering process is performed only once, at the start up of the system, in a situation in which the keys are randomly placed on the peers. However, this very unfortunate situation will never occur again: in a steady situation, in which most of the keys are already ordered, the correct relocation of new keys will be much faster, as Section 5.3 will show.

A search request is issued by a peer to find as many keys as possible that have a specified value, and therefore belong

to a given class. The goal of the discovery procedure is to drive a search message towards the peer whose centroid is the closest to the target key value. Indeed, in the steady situation, the values of the keys stored by a peer are always very close to the peer centroid. As a confirmation of this, Figure 8 shows 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[3]. This figure shows that the values of the large majority of the keys are very close to the respective peer centroids: the number of permitted key values is 1024, but the distance between the key value and the respective centroid is very rarely larger than 5.

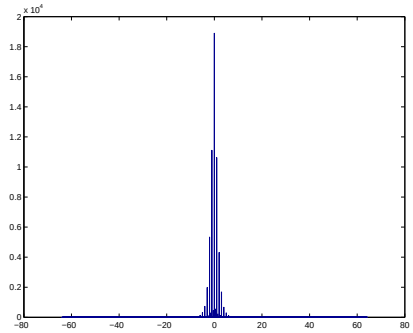

Figure 8. Distribution of keys with respect to the peer centroid, in a network with 4096 peers, and 1024 resource classes. For each key, the figure reports the relative distance between the key value and the centroid of the local peer.

The ordering of keys over the ring can be 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 described in Section 4.2, the destination peer is selected by making a proportion between the resource keys and the peer indexes. The destination peer is calculated as described in formulas (7) and (8): if the centroid of the destination peer is found to be closer to the target key than 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, the discovery procedure terminates, because with high probability the current peer is the one that stores the largest number of keys having the desired value.

If the keys are uniformly distributed - that is, if key values are assigned to resources in a completely random fashion - 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 [21] [4]. Figure 9 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. These results are obtained with a uniform distribution of keys and are relative to search requests issued in the steady situation. Here it is worth recalling that the average number of steps experienced in Chord is equal to $\frac{1}{2}\lg_2 N_p$ [21], but it is reduced to $\frac{1}{3}\lg_2 N_p$ in BiChord [15], in consequence of the presence of the reverse finger table. Figure 9 shows that the self-organizing reordering of keys achieved by Self-Chord agents does not worsen the performance of discovery requests: 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.

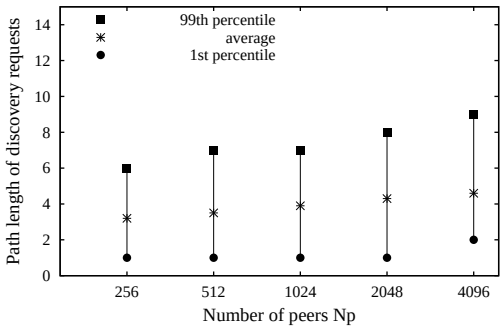

Figure 9. Path length of discovery requests with variable number of peers. The average, the 1st and the 99th percentile are reported.

Figure 10 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 that also these neighbor peers store a significative number of keys having the desired value. In Figure 10, 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 $(N_p * N_{res})/N_c$. 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, equal to 10 in these experiments, and N_c is the num-

[3]Here a distance is considered positive if the key value is higher than the centroid, in an arithmetic modulo N_c, negative in the other case.

[4]In Section 5.2 it will be shown that the discovery procedure is efficient even with a non-uniform distribution of keys.

ber of resource classes, which is equal to 1024. In conclusion, the discovery procedure successfully discovers nearly all the resources that have the desired value of the key.

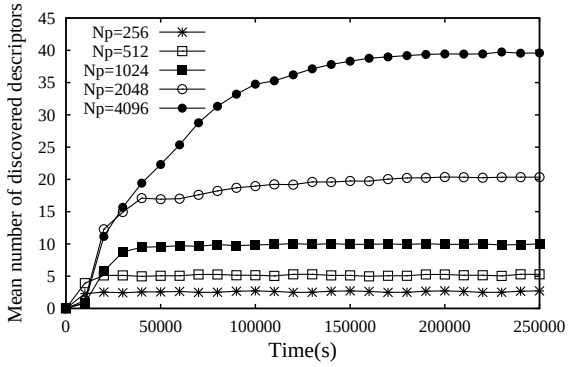

Figure 10. Average number of keys discovered by a query vs. time, with variable number of peers.

5.2 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 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, thus invaliding the assumption. Moreover, in Self-Chord a resource key can also 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. In the introductory section, it was mentioned that the self-organization of keys performed by Self-Chord agents allows the load to be fairly balanced among the peers, even in the case of non-uniform distribution. This advantageous characteristic is illustrated in this section.

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 $\frac{N_c}{2}$ is the most frequent value, whereas values 0 and $N_c - 1$ are the least frequent. Figure 11 shows the pdf of the triangular distribution that is obtained with these assumptions.

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

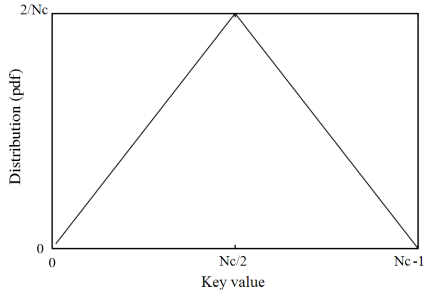

Figure 11. Example of a non-uniform distribution of keys: the triangular distribution. With the number of resource classes set to N_c, $\frac{N_c}{2}$ is the most frequent key value.

keys to the peers in a fair fashion, both with uniform and non-uniform distribution of keys. In the case of non uniform 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 of this 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 Figure 12, 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, $\frac{N_c}{2}$, which in this case is equal to 512, while fewer peers have centroids with values close to infrequent keys.

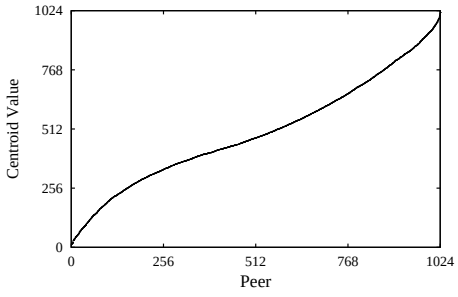

Figure 12. Centroid values of peers. The first peer on the x axis is the one with the lowest centroid value. The others are taken from the ring following the clockwise direction.

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 both with the uniform and the triangular distribution of keys, and are equal to 10, 2 and 22, respectively. Interestingly, the variability of the number of keys stored per peer is much lower than that experienced in Chord with a uniform distribution. For example, the 99th percentile calculated in Chord under the uniform assumption, and reported in [21], is about 50, compared to the value of 22 experienced in Self-Chord. Therefore, the work of agents in Self-Chord is capable of significantly improving the load balance also with a uniform distribution, and the advantage is much larger with a non-uniform distribution. It should be remarked here that this fair load balance is obtained in a totally decentralized and self-organizing fashion, while it would be very difficult to achieve with any centralized algorithm. This confirms the surprising efficacy of these very simple nature-inspired mechanisms, especially in a large distributed environment.

In Section 5.1 it was mentioned that the resource discovery algorithm estimates the index of the next peer to which a query message is forwarded, assuming a uniform distribution of keys. The discovery procedure could become longer with a non-uniform distribution, because 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 13 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, with a variable value of N_p. 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, whereas in the most unfortunate cases (evaluated through the 99th percentile) at most 2 more steps are required to successfully complete the discovery procedure. In all cases, the number of steps is logarithmic with respect to the number of peers, thus guaranteeing an excellent scalability behavior of the algorithm.

5.3 Dynamic Behavior

In Section 5.1, it was mentioned that the greatest part of the reordering work must be executed only once, when the algorithm is started for the first time. Once a correct reordering has been obtained, it can be kept with few relocations of keys. In particular, it is necessary to reorder the keys of the resources that are dynamically added to the network, for example by new or reconnecting peers. In a system like Chord, the keys must be immediately placed in the correct hosts: this can originate a high network and processing load if the system is highly dynamic, especially if many resources are published in a short interval of time.

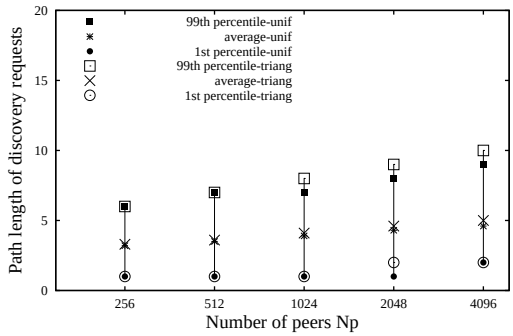

Figure 13. Path length of discovery requests: average, 1st and 99th percentile calculated with uniform and triangular distributions of keys and a variable value of N_p.

In Self-Chord, the load is invariant because a new or reconnecting peer does not need to perform any additional operation: the keys of the new resources will be picked by the agents that arrive at 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. It does not depend on the frequency of peer joinings and disconnections, but only on the probability that a reconnecting peer generates an agent, P_{gen}, and on the frequency of agent movements across the Grid, $1/T_{mov}$. In fact, L can be calculated by multiplying the overall number of agents 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 number of peers N_p, so obtaining the number of times per second that an agent arrives at a specific peer [5]:

$$L_p = \frac{N_a}{N_p \cdot T_{mov}} \approx \frac{P_{gen}}{T_{mov}} \qquad (12)$$

In the described scenario, the average value of T_{mov} is equal to 10 seconds, and P_{gen} is set to 1.0: therefore, each peer receives and processes about one agent every 10 seconds, which is an acceptable load, since take and leave operations are simple and quick. This result, obtained theoretically, has been fully confirmed by simulation data. Note that the processing load does not depend on other system parameters such as the network size, the average number of resources published by a node and so on, 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

[5] The simplification in formula (12) is obtained by using formula (1), in which T_{agent} is assumed to be approximately equal to T_{peer}

4.4). 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 by setting a counter on a new key and initializing its value to a small integer value, c. The agent that picks the key checks if the counter is greater than zero: if so, it jumps with the logarithmic mode and decrements the counter, otherwise it operates in the usual fashion and moves on the network according to the mode of the local peer. Therefore a new key is moved with the logarithmic mode at least c times, which assures that it can be rapidly placed in the correct peer, with the only condition that c is 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 c to an appropriate value.

The disconnection of a peer does not need additional operations of peers or agents. 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.

A set of specific experiments was performed to evaluate the dynamic behavior of Self-Chord: once the reordering process has reached a steady condition, a perturbation is generated by simulating the simultaneous arrival of a number of new peers in the system, each with 10 new resources on average. The initial number of peers N_p is set to 1024, but after 100,000 seconds, a number of new peers, specified as a percentage P_{join} of N_p, join the network.

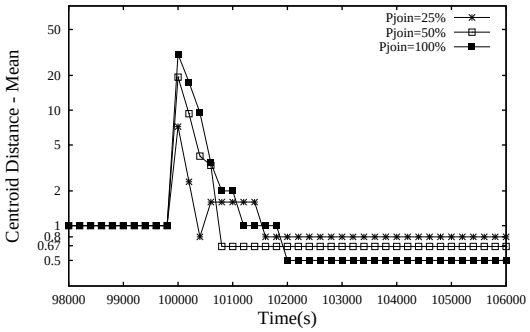

Figure 14. Average distance between two consecutive centroids. At the beginning, the values of N_p and N_c are set to 1024. At time 100,000 a percentage P_{join} of new peers join the network.

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 14 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%. 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 no more than 2000 seconds, about half an hour. 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 Figures 14 and 7 is interesting. While the agents take about 50,000 seconds to order the keys in a network with 1024 peers, if they start from scratch, they take only 2000 seconds to order the keys published by additional 1024 peers. The reason is that the new keys are inserted in a network already ordered, in which it is much easier to find their correct location by using the logarithmic approach. This result is important, because it proves that the Self-Chord algorithm 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 Figure 14, is easily managed by Self-Chord agents, and the key ordering is recovered very rapidly.

6 Conclusions

This paper presents Self-Chord, a "self-structured" P2P system built in accordance with a bio-inspired algorithm. In Self-Chord, a set of ant-inspired mobile agents move and reorder the resource keys in a ring of peers. Self-Chord aims to preserve the advantages of Chord, such as the logarithmic search time, but offers further profitable characteristics inherited by biological systems, such as self-organization, adaptivity, scalability and fast recovery from external perturbations. The efficiency and effectiveness of Self-Chord are confirmed by results obtained with an event-based simulation framework. Even if Self-Chord is here compared to Chord, the presented approach is extensible 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. Even in these cases, the self-organization and ordering of keys can be achieved with small modifications of the bio-inspired algorithm presented in this paper.

Acknowledgment

The authors would like to thank Emilio Leonardi, Politecnico di Torino, for his contribution in the definition of the model and Giandomenico Spezzano, ICAR-CNR, for his help in the analysis of bio-inspired algorithms.

References

[1] S. G. R. B. Anand Padmanabhan, Shaowen Wang. A self-organized grouping (sog) method for efficient grid resource discovery. In *Proc. of the 6th IEEE/ACM International Workshop on Grid Computing*, Seattle, Washington, USA, November 2005.

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

[3] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of the Second IEEE International Conference on Peer-to-Peer Computing P2P'02*, pages 33–40, Washington, DC, USA, 2002. IEEE Computer Society.

[4] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proc. of the 22 nd International Conference on Distributed Computing Systems ICDCS'02*, pages 15–22, Washington, DC, USA, 2002. IEEE Computer Society.

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

[6] 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.

[7] A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 35(3):373–384, 2005.

[8] A. S. Cheema, M. Muhammad, and I. Gupta. Peer-to-peer discovery of computational resources for grid applications. In *Proc. of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 179–185, Seattle, WA, USA, November 2005.

[9] 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.

[10] 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.

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

[12] M. Fukuda and D. Smith. UWAgents: A mobile agent system optimized for grid computing. In *Proc. of the 2006 International Conference on Grid Computing & Applications*, pages 107–113, Las Vegas, Nevada, USA, June 2006.

[13] P. Grasse'. La reconstruction du nid et les coordinations inter-individuelles chez belicositermes natalensis et cubitermes sp. la thorie de la stigmergie : Essai d'interprtation du comportement des termites constructeurs. *Insectes Sociaux*, (6):41–84, 1959.

[14] 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*. Kluwer Publishing, 2003.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] K. Sycara. Multiagent systems. *Artificial Intelligence Magazine*, 10(2):79–93, 1998.

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

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