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

Agostino Forestiero  
forestiero@icar.cnr.it

Carlo Mastroianni  
mastroianni@icar.cnr.it

ICAR-CNR  
Institute for High Performance Computing and Networks  
Cosenza, Italy

Michela Meo  
michela.meo@polito.it

Politecnico di Torino  
Torino, Italy

Structured P2P and Self-Organization

- Can Structured P2P be Self-Organizing?
  - So far, self-organizing and adaptive properties have always been associated to unstructured P2P systems. Structured systems are generally considered too rigid to present self-* behaviours.
  - Self-Chord tries to confute this belief. It exploits the structured topology of Chord, but features self-organizing mechanisms for key ordering and discovery.

- How are self-organizing properties obtained?
  - By using mobile agents, which travel the Chord ring and order resource keys.
  - Mobile agent operations are inspired by the behaviour of ants, in accordance with the swarm intelligence paradigm.

- What are the benefits w.r.t. classical structured systems?
  - Essentially three:
    - Peer indexes and resource keys are decoupled, which opens the possibility to give a semantic meaning to keys and perform class (or range) queries.
    - A better load balance among the peers can be obtained.
    - Dynamic properties (e.g., management of joining nodes) are improved.
Quick resume of Chord

- In Chord each **peer** is assigned an **m bit code**.
- Chord manages a logical circle: the peers are ordered with respect to their codes.
- Each **resource** is also assigned a **key**, generated by a hash function. Keys have **m bits**, like peer codes.
- Each key is assigned to the peer having a code equal to the key. If no such peer is present on the network, the key is assigned to the **successor** peer, i.e., the first peer available.
- Each peer $K$ manages a **finger table** that contains the addresses of other $m$ peers:
  - The $i$-th record ($i=0..m-1$) of the finger table of $K$ points to the peer whose code is obtained by summing $2^i$ to $K$ or, if that peer is not present, to its successor.
  - The finger tables allow discovery operations to be executed in **log time**.

Chord: an example
Key concepts of Self-Chord

- **Basic features**
  - Peer indexes and resource keys are decoupled and defined on different spaces:
    - Peer indexes are in the range $0..2^{ln-1}$
    - Resource keys are in the range $0..Nc-1$
  - A resource key can be given any meaning, also semantic. Moreover, resources can be categorized into $Nc$ classes
  - There is no predefined association among keys and peers, differently from Chord
  - But keys must be ordered over the ring to let discovery operations be executed in log time

- **Key ideas**
  - For each peer we define its centroid. This is the value that minimizes the average distance between itself and the keys stored in the local region
    - The centroid is an indication about the key values that are stored in a given region of the ring
  - The mobile agents try to bring each key to the peer whose centroid is the closest to the key
  - This simple strategy assures a quick, precise and robust ordering of the keys!

More about the centroids...

- **Two examples (with Nc=64):**
  - If the keys stored in the local region are {000100, 000110, 001000} or, in decimal notation, {4, 6, 8}, the centroid of the local peer is 6
  - If the keys stored in the local region are {000000, 111111}, or {0, 63}, the centroid is 63.5
  - The centroid is computed from the keys of three adjacent peers, not just the local peer. In other words, the keys stored in a peer concur to the computation of three successive centroids.
    - This is essential to guarantee the gradual ordering of centroids and keys
A Visual Introduction to Self-Chord

In this sample of Self-Chord ring...
- Peer indexes are defined over 6 bits
- 16 peers are actually connected on the ring
- Resource keys are defined over 3 bits
- Each peer publishes 10 keys on average, and their key values are assigned randomly

The figure shows a snapshot of the network in a steady situation

Notice that:
- There is no direct association among a peer index and the keys stored by the peer
- The peer centroids are ordered in clockwise direction
- The keys are also ordered (only 3 keys per peer are shown)
- Each key is stored in a peer whose centroid is very close to the key value

Key ordering is the base for resource discovery!

Operations of Self-Chord agents

- Each agent performs a few simple operations, cyclically:
  1) while it is not carrying any key, the agent hops from a peer to its predecessor or successor, depending on the agent being left-handed or right-handed;
  2) at any new peer, it decides whether or not to take a key out of the peer;
  3) after taking a key, the agent jumps to a new peer exploiting the current peer’s finger table
  4) at the new peer, the agent decides whether or not to leave the carried key.

- Operations (1-2) and (3-4) are repeated until the agent takes or leaves a key, respectively

- Take and leave operations depend on probability functions
Take Operation

- To decide whether or not to take a key, the agent evaluates the similarity of the key with the other keys stored in the local region:
  - The similarity function measures the similarity between the key \( r \) under evaluation and the centroid of the local peer, \( c \):
    \[
    f(r, c) = 1 - \frac{d(r, c)}{Nc/2}
    \]
    \( d(r, c) \) = distance between \( r \) and \( c \)
    \( Nc/2 \) = maximum distance
  - The value of \( f(r, c) \) ranges from 0 (minimum similarity) to 1 (maximum similarity)
  - The decision to perform the take operation is the result of a Bernoulli try. The take probability is defined to be inversely proportional to the value of \( f(r,c) \)
    \[
    P_{\text{take}} = \frac{k_t}{k_t + f(r,c)}
    \]
    \( k_t \) = parameter with value in the range \( [0,1] \)

Agent Jump

- The agent, after taking a key, tries to jump to the region of the ring where this key should be deposited
  - This is done in a few steps:
    - first, the distance of the key \( r \) from the local centroid \( c \) is calculated, in the space of resource keys:
    - then, this distance is mapped to a distance in the space of peer indexes, with the following proportion:
      \[
      \frac{r - c}{Nc} = \frac{P_d - P_s}{Nr}
      \]
      \( N_r \) = number of admissible key values
      \( N_c \) = number of admissible peer indexes
      \( P_s \) = this peer
      \( P_d \) = destination peer
      \[
      P_d = P_s + \frac{N_r}{N_c} (r - c)
      \]
    - finally, the agent jumps to the peer, indexed by the finger table, which is the closest to \( P_d \)
  - The centroid of this peer is likely to be much closer to the value of the target key
Leave Operation

- The probability of dropping a key $r$ must be directly proportional to the similarity between the key and the centroid $c$ of the new peer

\[ P_{\text{leave}} = \frac{f(r, c)}{k_1 + f(r, c)} \quad (0 < k_1 < 1) \]

- The leave operation is also the result of a Bernoulli try
  - if the key is dropped, the agent will resume to move “linearly” and try to take another key
  - otherwise, it will jump to another peer, as described before

- In an ordered ring, the agent will get to the correct peer in logarithmic time
- If the ring is not yet ordered, take and leave operations will contribute to reorder it!

Life cycle of Self-Chord agents

- Each peer, when joining the network, generates an agent with probability $P_{\text{gen}}$
- The agent is right-handed or left-handed with the same probability:
- Each peer estimates its average connection time. The life time of the agent is set to this time
  - In this way, the agents die, on average, when the peers that have generated them disconnect from the network
- This strategy assures that a stable number of agents circulate in the network at any time, despite the fact that peers disconnect and reconnect, and that agents are generated and die
- The average number of agents $N_a$ is correlated with the average number of active peers, $N_p$, and can be tuned through the parameter $P_{\text{gen}}$:

\[ N_a \approx N_p \cdot P_{\text{gen}} \]
Performance Evaluation

- Simulation experiments have been performed to assess:
  1) the correct reordering of centroids and keys
  2) the path length of discovery operations
  3) the load distribution
  4) the dynamic behavior

- The simulation scenario is the following:
  - Number of connected peers \( N_p \): 256 to 4096
  - Resource keys defined over 10 bits (\( N_c = 1024 \) classes)
  - Peer indexes defined over 16 bits
  - \( P_{\text{gen}} = 1 \): each peer generates one agent
  - Each peer publishes 10 resources on average
  - Peer connection time: 5 hours on average
  - The time unit is defined as the average time between two successive movements of an agent (for example, time unit=100 ms)

How to evaluate the reordering of keys

- In an ordered ring, the peer centroids are ordered and uniformly spaced
- This can be assessed by computing the average and the standard deviation of the distance between two consecutive centroids:
  - The average must be equal to \( \frac{N_c}{N_p} \), which means that the centroid values are ordered and cover exactly one circle
  - The standard deviation must be low, meaning that the centroid values are uniformly spaced
  - Example: in the figure below, \( N_p = 16, N_c = 8 \), and the average centroid distance is actually \( \frac{N_c}{N_p} = 0.5 \)
  - The centroids are not spaced perfectly because the ring is very small and there is a strong discretization: the larger the network, the better the reorganization accuracy!
Average centroid distance

- Average centroid distance vs. time, starting from a completely disordered network.
- Curves are obtained for different network sizes \((N_p)\). \(N_c\) is fixed to 1024.

![Graph showing average centroid distance vs. time for different network sizes.](image)

- It is observed that:
  - the centroid distance always converges to \(N_c/N_p\) (and the standard deviation, not shown, decreases remarkably).
  - the time to converge increases with the network size.
  - however, convergence is much faster if the network grows gradually (later).

Distribution of keys

- Distribution of the distance between the keys and the local centroids.
- In this tests, \(N_p=4096\), \(N_c=1024\).

![Graph showing distribution of key distances.](image)

- the distance between a key and the centroid of the local peer is always very low (in an unordered network, the average distance would be \(N_c/4 = 256\)).
- therefore, a key can be looked up by searching the peer with a similar centroid value!
- the distribution is even narrower as the network size increases, because the impact of discretization is lower.
Discovery operation

- A discovery procedure is issued to search for the keys with a target value.
- The search message jumps to the peer whose centroid is estimated to be the closest to the target key.
- An example, assume that the red peer, with index 15 and centroid 4, issues a request for the target key $r = 000$.
- In this example, $c = 4$, $r = 0$, $N_r = 64$ (number of admissible peer values), $N_c = 8$ (number of admissible key values).

![Diagram showing the discovery process]

1) $P_d = P_b + \frac{N_r}{N_c} (r - c)$
2) $P_d = 15 + \frac{64}{8} = 15 + 8 = 47$
3) The search message follows the pointer to the peer whose index is 47 (101111)
4) If needed (not in this case), the procedure is repeated.
5) In this example, the target keys are found in one step, in peer 101111 and in the adjacent peers.

In general, the procedure requires $\log(N_p)$ steps, as in Chord.

Path length of search messages

- Average path length, and the 1st and 99th percentiles.
- In these tests, $N_c = 1024$, and the network size is varied.

![Graph showing path length and percentiles]

- The average path length is very close to $\log(N_p)/3$, which is the expected value for a ring with direct and reverse finger tables.
- The 99th percentile proves that the discovery time is logarithmic also in the most unfortunate cases!
Average number of discovered results

- A query can find several resources having a specified target key
- The figure shows the average number of discovered keys while the network is being reordered

\[
\frac{N_p \cdot N_{res}}{N_c} \approx 4096 \cdot 10 \quad \text{so} \quad N_c = 1024
\]

- in a network with \( N_p \) peers, \( N_c \) resource classes and \( N_{res} \) resources published by a peer, the number of resources of a given class is \( (N_p \cdot N_{res})/N_c \) on average
- the steady value of the index is comparable to the theoretical value: a query discovers nearly all the desired resources!

Non uniform distribution of key popularity

- The discovery procedure implicitly assumes that all the keys have comparable popularities
- What happens if it is not the case? For example, if keys are assigned with a triangular distribution

\[
\frac{2}{N_c} \quad \text{Distribution (pdf)}
\]

- with this distribution \( N_c/2 \) is the most frequent key: a large number of resources are assigned this or similar keys
Discovery with non uniform distribution

- Average path length and 1st and 99th percentiles, with uniform and triangular distributions of keys

> the results prove that the non uniform distribution has a very small impact
  ✓ the average path length increases very slightly
  ✓ the 99th percentile increases by at most one step

> in the non uniform case, the first jump can be erroneous, but the error is soon compensated by the next jumps, due to the logarithmic approach

Load Balance

- 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
  > They are equal to 10, 2 and 22, respectively.

- In Chord, the 99th percentile under the uniform assumption, is about 50, compared to the value of 22 of Self-Chord.

- With a non uniform distribution, an acceptable load balance can be maintained in Chord only by defining additional structures, such as virtual nodes.

- Self-Chord does not need any superstructure to achieve a fair load balance.
Dynamic behavior

- If a new peer joins the network, its keys will be automatically reordered by the agents, in log time.
- But what happens if several peers arrive simultaneously?
- In a network of 1024 peers, a percentage $P_{join}$ of new peers arrive simultaneously.

- When the new peers arrive, a perturbation is generated and the centroid distance is altered.
- In a short time, however, the agents reorder the network.
- Notice that:
  - the average centroid distance converges to the new value of $N_c/N_p$ ($N_p$ has been increased).
  - the time to reorder the keys of 1024 new peers is much shorter than the time needed to order a network from scratch.
- In conclusion, Self-Chord behaves very well when the network grows gradually, which is the ordinary case.

Self-Chord prototype

- A prototype of Self-Chord is available at [http://self-chord.icar.cnr.it](http://self-chord.icar.cnr.it)
- The prototype has a GUI that allows the user to create or join the network, publish resources, search for resources by name or by key.
- The Java source code is free (GNU licensed), so we hope you will use it and contribute if you like ☺
This is not the end...

1) We are developing an analytical framework that models the dynamics of the system through fluid-like differential equations

2) Bio-inspired algorithms can be usefully applied to other types of P2P structure