A Self-Organizing P2P System with Multi-Dimensional Structure

Presentation and Analysis of Self-CAN

Raffaele Giordanelli
giordanelli@icar.cnr.it

Carlo Mastroianni
mastroianni@icar.cnr.it

Michela Meo
michela.meo@polito.it

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

Politecnico di Torino
Torino, Italy
Structured P2P Systems

- **Benefits of Structured P2P Systems**
  - Modern P2P systems adopt **structured overlay**: rings, multi-dimensional grids, trees
  - Resource keys are assigned to peers using **hash functions**
    - *a resource discovery operation is transformed into a peer lookup operation, which can be performed in logarithmic time*
  - Structured P2P overlays and **Distributed Hash Tables** are also adopted in **Grid and Cloud Computing**. One major example: *Amazon Dynamo*

- **But there are also drawbacks...**
  - **Lack of self-organizing properties**. Structured systems are rigid.
    - *Difficulties to adapt to rapid changes, for example to node churning*
  - Management of **complex and range queries**. Due to the use of hash functions, the keys of similar resources are spread over distant peers
  - **Load balancing** is far from optimal. The peers that are assigned the most popular keys may be significantly more loaded than others.
Self-* Properties in Structured P2P Systems

- **How can self-organizing properties be obtained?**
  - We use **mobile agents**, which travel the overlay and order resource keys
  - Agent operations are inspired by the behaviour of **ant colonies**

- **What are the benefits vs. classical structured systems?**
  
  *Essentially three:*
  - Peer indexes and resource keys are **decoupled**, which opens the possibility to give a semantic meaning to keys, and helps the execution of class/range queries
  - **Better load balancing**
  - **Dynamic properties** are improved (e.g., management of joining and departing nodes)

- **What are the specific features of Self-CAN?**
  - Self-CAN exploits the **multi-dimensional overlay** of CAN, but adds self-* properties
  - Multi-dimensional values of resource keys can match **resource attributes**: e.g., the attributes of a host can express CPU power, RAM memory and type of operating system
  - Efficient execution of **multi-dimensional range queries**, essential in Grids and Clouds
Quick resume of CAN (1/2)

- In CAN each **resource** is assigned a **point** in a D-dimensional space.
- Each **peer** is assigned a **region** in the same space, and is required to manage the resource keys comprised in that region.
- Each peer is connected to **2D** neighbors, 2 for each dimension

Example of a 2-dimensional CAN overlay with 5 nodes
(from the CAN paper)
Quick resume of CAN (2/2)

- The search path follows connections among adjacent peers. If a D-dimensional space is partitioned among $N_p$ peers, the average path length is $\frac{1}{4} \cdot D \cdot N_p^{1/D}$.

- If the value of $D$ is logarithmic with respect to the number of peers $N_p$, other important indexes are logarithmic. For example, if $D \sim \frac{\log_2 N_p}{2}$:
  - the average length of the search path is $\frac{\log_2 N_p}{2}$
  - the number of a peer’s neighbors is $\log_2 N_p$

Example of a routing path in a 2-dimensional CAN overlay
(from the original CAN paper)
Key concepts of Self-CAN

- Self-CAN exploits the CAN overlay, but **decouples** resource keys and peer indexes.
  - key values may be associated with resource attributes

- The resource keys **self-organize** along the D-dimensional space while preserving the **sorting of key values along each dimension**
  - queries can reach a target key without knowing in advance the identity of the responsible peer
  - efficient management of **range and class queries**

- Key sorting is obtained through **pick** and **drop** operations of ant-inspired mobile agents.

- Keys are moved using **peer centroids** as a reference.
Peer centroids

- The centroid of a peer is the **D-dimensional vector** of key values that minimizes the distance from itself and the values of the keys stored in the local region.

- Example: \( D=2 \), and for each dimension keys assume values in the range \( \{0..15\} \). Resources are so categorized in 256 classes.

- The keys stored in the local region are: \((1,15)\), \((3,14)\), \((5,0)\), \((3,1)\).

- The centroid is \( C=(3,15.5) \), because for each dimension the centroid value minimizes the “circular” distance between itself and the keys.

- The objective of mobile agents is to pick and drop keys by comparing them to centroids. Some examples:
  
  - *the key \((1,15)\) should be moved towards a predecessor peer along the first dimension*
  
  - *the key \((3,1)\) should be moved towards a successor peer along the second dimension*
  
  - *an agent arriving at this peer with the key \((3,15)\) should drop it in the current peer*
Sorting of keys and peer centroids

- Pick and drop operations gradually sort keys and peer centroids over the D-dimensional structure.
- Once the ordering is achieved, it is easily maintained even when the environment changes, for example when peers connect/depart, or keys are added/modified.

A 2-DIMENSIONAL SELF-CAN NETWORK WITH 16 PEERS.
PEER CENTROIDS ARE SORTED ALONG BOTH DIMENSIONS.
Discovery procedure

- Every key is stored in a peer whose **centroid is very similar to the key**
- It is possible to search a key by following the **gradient of peer centroids** along the D dimensions.

**Diagram:**

Peer B stores key \((9,6)\)

The peer A starts a resource procedure to find key \((9,6)\).

The search message follows the centroid gradient and arrives at peer B, which most probably stores the desired key, because the centroid is very similar.
Operations of Self-CAN agents

- Each agent performs a few simple operations, cyclically:
  1) while it is not carrying any key, the agent hops from a peer to an adjacent one, chosen randomly;
  2) at any new peer, it decides whether or not to pick a key out of the peer;
  3) after taking a key, the agent jumps to a new adjacent peer, moving towards a region in which peer centroids are more similar to the key;
  4) at the new peer, the agent decides whether or not to drop the carried key.

- Pick and drop operations are executed on the basis of Bernoulli trials whose probability depends on the similarity between the key and the local centroid.
To decide whether or not to pick a key, the agent evaluates the similarity $f(k,c)$ between the key $k$ and the local centroid $c$:

$$f(k, c) = 1 - \frac{1}{D} \sum_{i=1}^{D} \frac{\Delta_i}{L_i/2}$$

- $\Delta_i = \text{distance between } k \text{ and } c \text{ over the } i\text{-th dimension}$
- $L_i/2 = \text{is the maximum distance over dimension } i$

The value of $f(k,c)$ ranges between 0 (minimum similarity) and 1 (maximum similarity).

The decision to perform the pick operation is the result of a Bernoulli trial. The pick probability for the trial is inversely proportional to the similarity $f(k,c)$.

$$P_{\text{pick}}(k) = \frac{\alpha_p}{\alpha_p + f(k, c)}$$

- $\alpha_p = \text{parameter with value between 0 and 1}$
**Jump and Drop**

- While carrying a key $k$, the agent selects the dimension along which the distance between $k$ and the centroid $c$ is the largest
  - if $k > c$ along this dimension, the agent jumps to the **successor** peer
  - if $k < c$ the agent jumps to the **predecessor** peer

- At any new peer, the agent tries to drop the key. The drop operation is the result of another Bernoulli trial

- The drop probability is **directly proportional** to the to $f(k, c)$, the similarity between the key and the new centroid

$$P_{drop}(k) = \frac{f(k, c)}{\alpha_d + f(k, c)}$$

- $\alpha_d$ = parameter with value between 0 and 1

- Pick and drop operations rapidly sort the keys over the multi-dimensional space!
Example of pick and drop operations

- An agent at peer A, with centroid (13.0, 14.0), evaluates the key (9,6)
- The similarity function is low, therefore the key is likely to be picked
- The key is brought towards the peer B having centroid (9.0, 6.5)
- The similarity is high, so the key is likely to be dropped here!

\[
P_{\text{drop}}(k) = \frac{f(k, c)}{\alpha_d + f(k, c)} = 0.85
\]

\[
P_{\text{pick}}(k) = \frac{\alpha_p}{\alpha_p + f(k, c)} = 0.64
\]
Performance Evaluation

- Simulation experiments have been performed to check:
  1) the sorting of centroids and keys
  2) the key clustering: are the keys stored in the same peer similar to each other? How much are they distant from local centroids?
  3) the performance of discovery operations (both punctual and range queries)
  4) the load balance

- Scenario:
  - Number of peers $N_p$: 256 to 4096
  - Number of resource classes $N_c$: 256 to 4096
  - Number of dimensions $D$: 2 to 6
  - $P_{gen} = 1$: each peer generates one agent
  - Each peer publishes 15 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 (in the experiments, time unit=5 s)
Curves are obtained for different network sizes \((N_p)\) and number of classes \((N_c)\).

If keys and centroids are correctly sorted, the average centroid distance tends to

\[
\frac{\sqrt{N_c}}{\sqrt{N_p}} = 1
\]

It is observed that:

- the centroid distance always converges to \(1\)
- the **time to convergence** increases with the network size, but is always reasonable
- In a steady condition, sorting is **rapidly recovered** after any environmental change
Homogeneity of keys

- The homogeneity function of a peer can assume values between 0 and 1, and higher values correspond to high degrees of clustering in the peer.

- After a rapid increase in the transient phase, the index stabilizes to values higher than 0.90.
- The index is hardly affected by the network size, which confirms the scalability of Self-CAN.
Distance between keys and local centroids

- Distribution of the distance between keys and local centroids. \( N_p=256, N_c=256 \)

- Before sorting the distribution of the distance is wide and centered on the value of 4
- After sorting the distance is between 0 and 1

- Consequence: A key can be discovered by directing queries towards the peer having a centroid value similar to the key
A query can find several resources having a specified target key (class query)

The figure shows the percentage of discovered keys with respect to the total number, while the network is being reordered.

- When keys are sorted, a class query can find practically **all the target keys**!
Path length of search messages

- **Average** length of the query path, and 1st and 99th percentiles

The average path length is about $\log(N_p)/2$: the mean feature of CAN, log search, is kept by Self-CAN.

- The **99th percentile** is about $\log(N_p)$, so the discovery time is logarithmic also in the most unfortunate cases!
Range Queries

- Two approaches to serve range queries: **sweep up** and **explosion**
  - The **sweep up** approach aims to reach every peer that may contain target keys included in the intervals of the range query.
  - The **explosion** approach aims to reach the core of the region, and then rapidly collect a sufficient number of target keys.

Giordanelli, Mastroianni, Meo: *A Self-Organizing P2P System with Multi-Dimensional Structure*
Load Balancing

- In this experiment the central peers store a larger number of **popular keys**
- Notation $N/M$ means: $N$ keys, $M$ of which belong to the most popular classes

<table>
<thead>
<tr>
<th>before sorting</th>
<th>after sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>18/0 18/0 19/0 18/0 20/0 17/0 22/0 21/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>20/0 19/0 20/0 22/0 19/0 20/0 18/0 18/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>19/0 19/0 23/0 24/10 23/11 21/0 20/0 23/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 50/50 50/50 20/0 20/0</td>
<td>25/0 21/0 26/13 27/27 23/28 26/8 25/0 22/0</td>
</tr>
<tr>
<td>20/0 20/0 50/50 50/50 20/0 20/0 20/0</td>
<td>25/0 21/0 26/7 27/27 23/28 28/8 23/0 23/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>21/0 24/0 19/0 26/16 25/17 20/0 20/0 19/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>18/0 23/0 19/0 24/0 24/0 18/0 23/0 22/0</td>
</tr>
<tr>
<td>20/0 20/0 20/0 20/0 20/0 20/0 20/0</td>
<td>20/0 19/0 24/0 25/0 24/0 21/0 21/0 17/0</td>
</tr>
</tbody>
</table>

- **Agent operation** distribute keys to peers in a **fair fashion**
- **Popular keys** are distributed over a larger number of peers
The prototype of Self-CAN is available at http://self-can.icar.cnr.it

The prototype has a GUI that allows the user to create or join a network, publish resources, search resources by name or by key

The Java code is available
Thank you for your attention!