Resource Location in P2P Systems

(extended abstract of the MSc dissertation)

João Pedro Fernandes Alveirinho
Mestrado em Engenharia de Redes de Comunicações
Instituto Superior Técnico

Advisor: Professor Luís Rodrigues

***Abstract*—Peer-to-peer (P2P) systems have emerged as a potential technology to build very-large distributed data and resource sharing systems. A key problem in these systems is the location of resources. Most approaches, address the scale challenge using either structured (DHTs) or unstructured overlay networks. Unfortunately, these solutions have a tradeoff between efficiency and flexibility. DHTs are more efficient but are specialized to address exact-match queries, whereas unstructured solutions, that typically rely in some sort of blind search mechanism, incur in additional overhead, but can address arbitrary complex queries. This thesis presents a novel self-organizing architecture to perform resource location. This architecure, that was named *Curiata*, combines structured and unstructured approaches to support flexible and efficient resource location. For instance, the architecture can be used to perform distributed resource allocation in cloud computing infrastructures. Experimental results extracted through simulation validate this design, and show that the proposed solution is able to offer a good recall, with small overhead and low latency.**

Keywords: *Resource Location, Self-Organizing Systems, Unstructured Overlays, Structured Overlays, Peer-to-Peer Systems.*

I. Introduction

Since the appearance of Napster[1] in 1999, P2P systems have been subject to intensive research and development efforts, both in academia and industry. Peer-to-peer file sharing systems such as Gnutella[2] and more recently BitTorrent[3] have had tremendous success. Unfortunately, these systems have been mainly used to illegally distribute copyrighted material. Still, several examples of legitimate uses of this technology also exist and can be found in, among others[4]. From the technological point of view, the potential of the technology to build extremely large-scale shared repositories makes it a very interesting research topic.

One of the main challenges in P2P systems is how to efficiently support resource location. Due to scalability and dependability issues, centralized solutions are not adequate. On the other hand, exhaustive search on all peers is also a non-scalable solution. Therefore, it comes as no surprise that resource location algorithms have been intensely studied, and many different solutions have been proposed.

II. Curiata

In order to improve the current state-of-the-art, a solution to the resource location in P2P system needs to address the following two key challenges:

i) Query Flexibility: Querying mechanisms should be flexible and allow rich and complex query languages. This would present an advantage when comparing to traditional DHT-based solutions which are typically only used for exact-match queries.

ii) Efficient Location of Resources: The proposed solution should be able to provide an efficient way to locate resources, in contrast with traditional unstructured solutions that typically rely on inefficient blind search mechanisms.

To answer these challenges the thesis proposes *Curiata*, a scalable and efficient resource location system that employs self-organizing techniques to integrate and combine the benefits of both structured and unstructured approaches. This approach supports flexible queries, like most unstructured solutions, while retaining the speed and efficiency provided by structured (DHT-based) solutions. The operation of *Curiata* is inspired by the organization of human societies. During the first two decades of the Roman Republic, the people were organized into units called *curia* of ethnic nature; the curia gathered into an assembly, the *comitia curiata*, for legislative, electoral, and judicial purposes, and where consuls had a special role. Similarly, in *Curiata*, peers self-organize in an unstructured overlay where nodes with similar shared resources establish neighbouring relations via a low-cost background process (the curia). Furthermore, nodes of each *curia* elect representatives to join a structured overlay (the *curiata*). Members of the structured overlay serve as contact points for other nodes with similar content. Thus, the structured layer is used to efficiently route queries towards regions of the unstructured layer which contain peers that share the type of resources being queried for. Then, the query is propagated amongst the members of the curia using unstructured techniques such as limited flooding or random walks. *Curiata* has a number of generic components that constitute the *Curiata* core architecture. Furthermore, some of these components can be specialized to optimize

Curiata's performance in each application scenario.

The components of a *Curiata* peer, are the following: i) The *resource index*, that describes the local resources available at the peer. ii) The *biased unstructured overlay* layer, that uses a distributed self-organizing protocol to ensure that peers establish neighboring relations with other peers with similar resources. iii) The *structured overlay* layer, which is only activated when a peer is elected as a *consul*. iv) The *consul election module*, that employs a local self-organizing protocol to select which peers belong to the structured layer. v) The *query routing* module, responsible for propagating and processing queries. In addition, some of these modules have specific methods that can be implemented in various ways to accomodate the requirements for different application scenarios.

A. Resource Index

In *Curiata*, it is assumed that resources can be classified in a set of categories. The resource index keeps a local record of all categories of the resources owned (locally) by the peer as well as (if applicable) the number of resources available for each of these categories. This information is used to identify which peers have similar resources. The classification scheme is orthogonal to *Curiata*. For instance, a distributed library of computer science papers could use the ACM Computing Classification System to classify the content. A distributed repository of music could extract the categories required to classify content from the most used tags used in popular applications such as "Last.fm" (http://last.fm).

B. Unstructured Layer

The unstructured layer organizes all peers that have available resources in a biased unstructured overlay network. More specifically, peers run a self-organizing distributed algorithm, to adapt the overlay topology according to the resources available at each node. The used algorithm is a specialized version of X-BOT [5], adapted to meet the requirements of *Curiata*.

X-BOT biases the unstructured overlay topology so that each node becomes neighbour with other nodes that have similar available resources. The rationale for this strategy is to be able to efficiently process queries by limiting the relevant areas of the unstructured overlay where nodes owning resources relevant for a query are located. For this, each node keeps two types of neighbors, denoted *active* and *passive*. The set of active neighbors defines the overlay that is used to propagate queries. Passive neighbours are used for exploring the network and finding other peers with similar interests. The passive view is updated by a periodic and random shuffle process similar to the one described in [6]. The active view is updated through the coordination procedure introduced by X-BOT, using a strategy specifically designed for *Curiata*. The the active view/unstructured layer update routine can be found in Listing. 1.

```
if(!active-View.has-Enough-Neighbours())
{
  //Establish network connectivity
  active-view.get-More-Neighbours();
}
else
{
 //Test if the active view is optimized
 //Instance Dependent!
  if(!active-View.is-Fully-optimized())
{
  //Trade old for ''better'' neighbours
  //Instance Dependent!
  active-View.optimize();
}
return;
```

Listing 1. Unstructured Layer/Active View Maintenance Routine

C. Structured Layer and Consul Election

The unstructured layer is able to organize itself so that it promotes neighboring relations among nodes that have similar resources (*i.e.*, resources fall into the same categories). Therefore, when searching for resources, as soon as one finds a peer that owns resources from the desired category, one can expect to find other nodes with similar categories in the overlay vicinity. The purpose of the structured layer is to facilitate the first step of the resource location procedure.

For this purpose, a fraction of the nodes that belong to the unstructured overlay also join a *DHT*. These nodes are elected to represent a category owned by nodes in each region of the unstructured overlay (or curia) and are referred to as *regional consuls* or *regional contacts*. A c-consul joins the DHT with an identifier constructed by assigning *hash(node_id)* to the s less significant bits and *hash(c)* to the remaining (most-significant) bits. Thus, the structured overlay layer acts as an assembly of representatives of the different regions in the unstructured space: its purpose is to efficiently route a query to regions of the unstructured overlay where the searched resources are likely to be located.

A c-consul node uses the unstructured overlay to periodically send a beacon to nodes in its r-vicinity. Nodes that receive the beacon, abstain from competing to become a c-consul. For higher values of r there'll be larger regions and a smaller number of consuls in the DHT. On the contrary, lower values of r will lead to the opposite scenario. The Structured Layer maintenance routine is in Listing 2.

```
if(this.is-Consul()){    // Test if node is already consul
  Category cat = this.get-Representing-Category();
  this.send-Beacon(cat,r);
  return;}
// Set of categories for which the node can be a consul
// Instance Dependent!
Array[Category] Categories
 = this.get-Categories-Node-Can-Represent();

for (Category ''c'' in Categories){
//Test if node has received a beacon for ''c''
  if(!this.has-Received-Beacon(''c''))
  { //Try to become a ''c''-consul
    this.compete-For-Election(''c'',r);
```

```
    return; }
}
//Terminate if categories already have a consul
return;
```

Listing 2. Structured Layer Maintenance Routine

If a node that belongs to a category c, that it can represent and, does not receive any beacons (within a time interval t), it competes with other potential candidates to become a c-consul. A node that is elected to be a c-consul, can use a consul of another category c' as a contact point for joining the DHT or may perform a random walk in the unstructured overlay to find a node that has information concerning any other regional *consul*. In addition, each node may only be the c-consul for a single category. A node that already is a c-consul does not compete to become the regional contact for any of the other categories.

D. Query Routing

Curiata does not constrain the format, nor the language, of the queries. There is only one requirement: from the query, it should be possible to extract the set $\mathcal{Q}$ of categories that match the query. For instance, if a query searches for a music by Aldina Duarte, one should be able to extract categories such as *world music*, *fado*, and *Portugal*[1].

Thus, the query routing module is reponsible for routing queries towards regions of the unstructured layer which contain peers whose categories match the ones defined in the query. This involves the partipation of nodes in the unstructured layer - forwarding the query within the regions that match such categories - and nodes in the structured layer, that route queries towards those regions.

To avoid flooding the network with query messages, the idea is to disseminate and process each query with some approximate message cost k. In the current prototype the value of k is static and defined offline. However, k could be dynamically adjusted to match the estimated rarity of the searched resource (for instance, based on the results returned by previous searches). A query is disseminated as follows:

- First, the query is routed to a member of the DHT.
- Then a copy of the query is routed to each category $c \in \mathcal{Q}$ using the DHT. Each copy is received by a c-consul for that category. To promote load balancing, for each category c, the query is routed to an identifier composed by *hash(c)*$||\{s$ random bits$\}$. This ensures that different queries are injected into the unstructured layer via different representatives of that category.
- Each c-consul starts a random walk of length $\frac{k}{|\mathcal{Q}|}$ in its vicinity. In each hop in the random walk, the query is preferably forwarded to a node that has not yet received the query These random walks are biased and are preferably forwarded to a neighbor that owns resources of the category c associated with the random walk.

[1]These categories were extracted from the user tags associated with this artist in Last FM.

- Each node visited by the random walks (including the c-consuls) executes the query locally and checks if it satisfies the query. In affirmative case, it adds to the random walk its own identifier.
- Finally, when a random walk reaches the maximum number of hops, it returns to the source all nodes matching the query that were visited by the random walk.

The total message cost of a query is $\frac{k}{|\mathcal{Q}|} \times |\mathcal{Q}|$ plus the cost of reaching a DHT member from the source (typically 1), plus the number of hops in the DHT required to reach the c-consuls (which is logarithmic with the number of peers in the DHT).

The query dissemination process may be optimized by having nodes avoiding to route the query to consuls of categories the originator of the query already owns. In this case, the originator of the query can use such neighbors as representative of these categories and having them initiate the random walk.

III. Instantiating the Architecture

Two possible instantiations for the *Curiata* arquitecture are now presented. For both scenarios, the *Curiata* core architecture remains unchanged; only the methods that are instance dependent are refined. In the first scenario the *Curiata* architecture is supports resource allocation in cloud computing infrastructures. The second scenario addresses resource location in P2P file sharing systems.

A. Resource Allocation in Cloud Infrastructures

In cloud computing infrastructures, outside users allocate resources in the cloud in order to store and execute their applications or services. However, inherent to the cloud computing paradigm is the ability to adjust or deploy, on demand, services to face dynamic changes on the workload. This brings upon the need for an infrastructure to keep track of the resource allocation in the cloud and to efficiently locate the resources required to launch/redimension a new/running service or application.

1) Resource Index: In this scenario, every resource available at each node in the cloud, which can be allocated by clients, is classified into some category from a finite set of categories defined a priori; although the number of categories can be arbitrarily high.

For instance, it is possible to classify nodes according to the amount of available memory as *memXS* (*1.7*Gb), *memS* (*7.5*Gb), *memM* (*15*Gb), etc. Also, nodes can be classified according to their architecture in the categories *32*bits or *64*bits. A similar approach can be used to classify other types of resources, such as disk space, number of cores, physical location, etc.

When characterizing resources that can take a discrete value from a possibly large range, the definition of categories simplifies the task of defining which nodes are more similar. In these cases each category represents an interval *I* of values

for that resource. Each peer in the system that shares that resource and has a value *v* for it, will fall into a category *c* so that $v \in I$.

2) Unstructured Layer: In this particular scenario, because each category owned by a node is considered to have the same importance for that node, our X-BOT proximity metric when applied to a pair of nodes returns a distance value that depends on the number of resource categories shared by those nodes. This means that, the returned value is zero if both nodes share exactly the same set of resource categories, and increases by a fixed amount for each category owned by only one of the nodes.

Therefore, X-BOT operates by exchanging overlay links, so that for each node *n* it maximizes the number of neighbors of *n* that share the same set of resource categories.

For instance, consider nodes *a*,*b*,*c* and *d* that belong to the following sets of category respectively:

- Node $a \in \{$*memXS*; *64*-bit; diskSpaceXL$\}$
- Node $b \in \{$*memS*; *64*-bit; diskSpaceXL$\}$
- Node $c \in \{$*memXS*; *64*-bit; diskSpaceXL$\}$
- Node $d \in \{$*memM*; *32*-bit; diskSpaceXL$\}$

The distance *d* between the each of the nodes is the following: *d(a,b) = 1*; *d(a,c) = 0*; *d(a,d) = 2*; *d(b,c) = 1*; *d(b,d) = 2*; *d(c,d) = 2*.

This means that in order for a node's active view to be fully optimized, all neighbours that belong to a node's active view need to have a distance of *0* to that node. Therefore, the *is-Fully-Optmized()* method only returns true when all the distances in a node's active view are *0*. Furthermore, in each optimization step, nodes try to gather peers in their active views so that their distance is as little as possible.In fact, each time this method is executed, the current node *n* trys to exchange a node *n'* in its active view with the highest distance to himself for a node *n"* from its passive with the lowest distance to himself if *d(n,n")<d(n,n')*.

3) Structured Layer and Consul Election: In this scenario, nodes can compete to become *consuls* in any category *c* to which they belong. Therefore, for this scenario, the *get-Categories-Node-Can-Represent()* method returns all the categories to which a node belongs. This is so because all categories are considered equally important for a node and therefore the node can represent any of them in the Structured Layer.

4) Resource Allocation: In response to a query, the source receives a list of at most k nodes that match the query. Some of these nodes serve as *regional consuls* and others do not. However, in this particular scenario, because resources are dynamic (i.e. they may change often overtime), to promote DHT stability, resources should be preferably allocated from nodes that are not *consuls*.

Resource allocation itself is application dependent and orthogonal to the *Curiata* architecture. For instance, a new service may be deployed on the selected nodes or be expanded to use additional resources available in the cloud.

When resources are allocated, the categories owned by a node may change. In response, X-BOT will iteratively move the node to another region of the unstructured overlay.

B. Resource Location in File Sharing Systems

In peer-to-peer file sharing systems users share different content with each other. From audio or video files, documents and e-books, to applications and computer games, a large variety of content is shared within these systems. Setting aside the legal issues sometimes associated with file-sharing systems, the truth is that they have had tremendous success and a definite impact on internet users around the world. One of the key challenges for these systems lies on how to efficiently locate the resources (files) that users are looking for.

1) Resource Index: As mentioned earlier in *Curiata*'s core architecture description, the resource index keeps a local record of all the categories of the resources owned by a peer, as well as the number of resources available for each of these categories. However, to accommodate the diversity in both shared content and quantity that characterizes users in a P2P file sharing system, the resource index component also keeps, for each category c of the resources owned by a peer, the fraction of resources of that peer that fall in that category.

This value is denoted by $frac_c$. For instance, consider a node that stores computer science papers, and half of these papers fit in the *self-stabilizing system* category. Then, $frac_{sss} = 0.5$. Also, all the categories are sorted by *frac* values. The top t categories are used to define the neighboring relations at the unstructured layer.

2) Unstructured Layer: For this scenario, the curia layer divides the active view in t slices, where t is the same configuration parameter that is used to select the t top categories with larger fraction of local resources. Each of these slices is devoted to one of the t top categories. The slice for category c has a dimension $slice_c$ in the interval $\lceil smin, d \cdot \text{frac}_c \rceil$. Where $smin$ is the minimum size of a slice, and is set as $\frac{d}{2t}$. For instance, consider a system where the curia layer is configured to select neighbors according to the top 5 categories ($t = 5$). Consider a peer p such that the top 5 categories $(c1, \ldots, c5)$ have the following associated fractions: $(0.5, 0.2, 0.1, 0.1, 0.1)$. If peer p had a degree $d = 20$, its active view would be sliced as follows: $(10, 4, 2, 2, 2)$. Considering the slicing of the active view as described above, a node uses the following steps to bias its neighbors:

i) The first concern of a peer when it joins the unstructured overlay is to fill its active view, regardless of the similarity of its potential neighbors. This step aims at ensuring network connectivity.

ii) After filling the active view, the next priority for the peer is to have neighbors that belong to his top categories.

(i.e, each slice contains at least one neighbor of that category).

iii) Then, the peer tries to exchange neighbors so that each slice is filled with neighbours that have resources of the corresponding category.

iv) Finally, as soon as this last criteria is achieved the peer stops executing the self-organizing algorithm and maintains its neighbors unchanged.

This slicing of the active view leads to a different implementation of the *is-Fully-Optimized()*. In fact, the active view is only considered to be fully optimized if all peers in the active view belong to at least one category from the node's top categories and if each slice is filled with peers that belong to that category. In addition, the *optimize()* method follows two steps. In the first step, the node attempts to trade neighbours that belong to neither of its categories for nodes that belong to at least one of his categories. In the second step, the node trys to gather enough nodes in each category so that the active view respects the pre-defined slice sizes. During this process, when a node finds a category *c* that lacks neighbours in the active view, if the active view still has some open slots, the node simply trys to gather one more neighbour from its passive-view that belongs to *c*. Otherwise, if the active view is completely full, the node trys to find a category *c2* that has a number of neighbours higher than its slice size and then trys to replace a neighbour from *c2* (that does not also belong to *c*) for a new neighbour from its passive view that belongs to *c*.

3) Structured Layer and Consul Election: In this scenario, nodes can only compete to become c-consuls for the category c that is their topmost category (i.e. the category with the highest fraction in a node's active view). Therefore, for this scenario, the *get-Categories-Node-Can-Represent()* method returns only the node's topmost category. This policy attempts to ensure that a consul for c has enough neighbours that belong c not to run the risk of receiving a query for c and not being able to forward it because all its neighbours that belong to c have suddenly failed. In addition, regarding the query routing procedure, when trying to locate rare resources, it could be important to start several random-walks, for each category *c* that the query targets, in the unstructured layes instead of only a single one. Having nodes only being able to represent their topmost category, effectively makes this feasible as they will have more neighbours to initiate these random-walks.

IV. Evaluation

The performance of the proposed solution was evaluated by simulation using the Peersim[7] simulator. Evaluation results for both the cloud computing infrastructure scenario (presented in III-A) and also for the peer-to-peer file sharing system scenario (presented in III-B) are now provided.

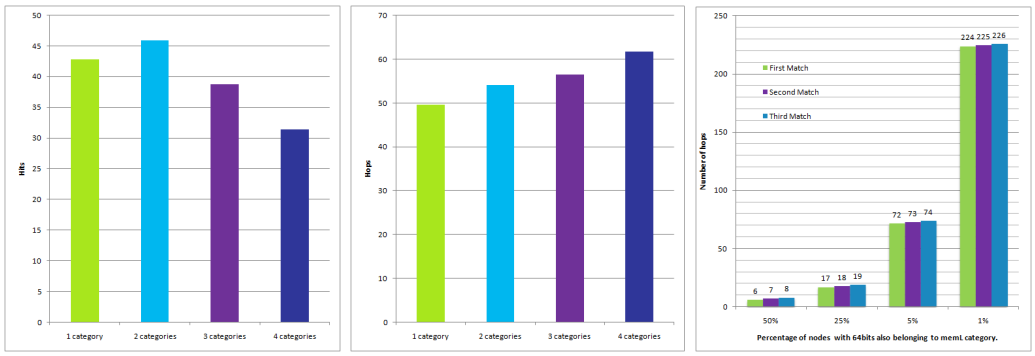
A. Cloud Computing Infrastructure

This scenario was based on the pre-configured instances provided by Amazon Web Services. In the experimental setup, 17 distinct resource categories were considered (5 categories for different sizes of each resource type – CPU, Memory, and Disk – and two categories for the CPU architecture – 32 and 64 bits), each node belongs to 4 of these categories (one per resource type) and maintains 30 unstructured overlay neighbors ($d = 30$). Finally, the radius to which nodes announce their presence in the DHT was set to 3 hops ($r = 3$). Each physical host, in the simulated cloud computing infrastructure, is assumed to own twice the resources of the most powerful pre-configured virtual machine used by the Amazon Web Services.

The system was evaluated for two types of queries. In the first experimental setting approximately 20.000 virtual machines were allocated with random configurations (from the ones employed by Amazon) over 10.000 physical nodes in order to promote the heterogeneity of available resources among nodes. Then, 4.576 queries were executed initiated from random nodes in the system, where trying to locate a machine with free resources in 1, 2, 3, and 4 different categories (1.144 queries of each type were executed). The k parameter was set (associated with the cost of performing random walks) to 48, therefore 48 nodes is the maximum number of nodes that can be returned by a query.

The goal was to evaluate the number of nodes returned by each query (*i.e.* number of hits) and the cost, in number of messages, of disseminating queries targeting different number of categories. Results are reported in Figures 1(a) and 1(b) respectively. On average, queries are able to return a large set of answers. The number of nodes returned by queries mostly decreases with the increase of associated categories. This is expected, as a smaller number of nodes in the system have adequate resources to reply to such queries. However, even when performing queries for 4 resource categories, our approach is able to find more than 30 nodes that fit the requirements. Interestingly a query for a single resource category returns less elements than a query that targets 2 categories. This likely happens because, in such scenario only a single random walk is employed which can transverse, and exit, the overlay region biased for that category, being unable to locate additional valid answers afterwards.

Figure 1(b) reports the message cost for disseminating each type of query. As expected the cost in number of messages slightly increases as the number of categories rises. This happens due to the additional cost of forwarding one message through the DHT for each query category. Notice however, that the cost does not rise linearly, as the cost of performing random walks is maintained constant ($k = 48$). Moreover, notice that, even when performing a query for 4 categories, the additional cost (*i.e.* the number of messages



(a) Number of Hits for Random Queries (b) Message cost for Random Queries (c) Hops for Rare Intersections

Figure 1. Resource Allocation

above 48) is, on average, below 15 additional messages.

Experiences have also been performed in an additional setting where queries target two resource categories being each of these categories very popular individually, but where the intersection of both is rare. Notice that this is a scenario that presents a relevant challenge to our system, as the DHT will route queries to nodes belonging to each of the categories individually, starting random walks in regions of the unstructured overlay network that are biased for each category independently.

To this end, the system was configured to have half of the computational nodes belonging to category *64bits* (*i.e.* 5.000 nodes) and a varying fraction of these nodes was configured to also have the category *memL*. 5.000 individual queries were then issued for each tested configuration. Only a single query was routed in the DHT to a representative of the *64bits* category, as this is the worst entry point for queries in this scenario. The number of hops required to find to first, second, and third nodes that belong to both categories were then measured.

Results are depicted in Figure 1(c). As expected, as the percentage of nodes belonging to *64bits* that also belong to *memL* diminishes, the random walk requires additional hops to find the region in the unstructured overlay where nodes belonging to both categories are clustered. This happens because the queries are injected into the unstructured layer at representatives of a single category therefore it is possible that the region of the unstructured overlay where the query is injected is only biased for *64bits*. Such limitation can however be easily circumvented, by allowing representatives of multiple categories to emerge. This question will be addressed as future work. Notice however that the biasing of the unstructured overlay network allows our system to find the second and third hit for the query in the hops immediately after locating the first hit.

B. P2P File Sharing Systems

In order, to demonstrate the importance of each module in the *Curiata* architecture, the performance of the following four different sub-architectures was compared:

Random walks on a Random Topology (RWRT): This corresponds to a system that uses only a random unstructured overlay, where no self-organizing background process is employed. In this architecture, there is no DHT and peers are not defined by the similarity of their resources.

Biased Random Walks on a Random Topology (BWRT): This corresponds to a system that uses uses a random unstructured overlay, where no biasing is applied. However, in this architecture, queries (*i.e.*, disseminated through random walks) are guided using a mechanism similar to the one used in *Curiata*.

Biased Random Walks on a Biased Topology (BWBT): This corresponds to a system that uses an unstructured overlay that is biased by the same self-organizing background process employed in *Curiata*. In this architecture the queries are also guided. However no DHT is used to route queries.

Curiata: This implements the full architecture as described in section III-B. In this architecture, peers self-organize to bias their neighbors according to the similarity of their resources and *consuls* are elected and organize themselves in a DHT.

1) Queries: Each query is implemented using a single k-length guided random walk with $k = 128$ and $k = 256$. The length of the random walk is counted from the peer that originates the query. Thus, all hops are considered, including the (non-random) hops required to reach the nearest consul, and then to route the query in the DHT until a representative of the searched category is found.

To simplify the analysis of the results, each query in our simulation searches for a single resource. Since each resource is associated with a single category, this category is used to guide the query, both in the BWRT architecture (to guide the walk in the unstructured layer), in the BWBT architecture and in the full *Curiata* system (both in the

unstructured layer and when routed in the DHT). Note that the fact that resources have unique identifiers is an artifact of the simulation. As noted before, *Curiata* is aimed at a system where users perform complex queries. Thus, a search for a specific resource simulates any complex query that is satisfied only by the resource with that identifier.

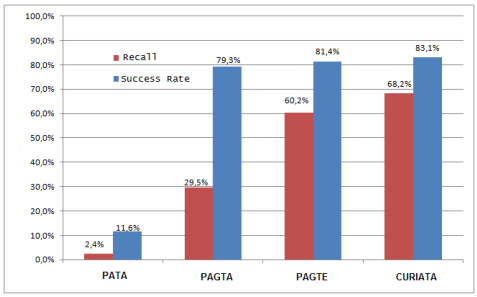
2) Metrics: Most of the experiments executed in this scenario use the following metrics:

- *Success rate:* The percentage of queries that found at least one copy of a given resource.
- *Recall rate:* The percentage of copies of a given resource that are found by a query (with regard to the total number of copies of that resource in the system).
- *Latency:* The number of hops required to find x copies of the resource. In most scenarios latency values to find 1, 2, and 3 copies of the resource were provided.

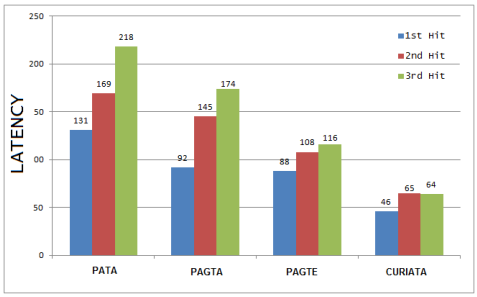
3) Experimental Setup A: For the first experiments a network of 10.000 peers was configured and 11 resource categories. Each node in the system is assigned 1 category out of the 11. Resources in the network are associated with a single category and have a unique identifier. Resources of each category c are randomly allocated to peers of that category, such that each resource exists in 5 distinct nodes.Each node was configured to have 20 neighbors in the unstructured overlay network. The radius of the regions for consul election in the curia layer is set to 2 and in this scenario approximately 200 peers out of the 10.000 are elected as consuls and join the structured layer. Furthermore, each category falls over one of 3 popularity ranks as follows: *Rare Categories:* categories with less than 100 peers (Categories G to K); *Intermediate Categories:* categories with 100 to 1.000 peers (Categories D to F); *Common Categories:* categories with more than 1.000 peers (Categories A to C). Category A is the most common category with 5.000 peers whilst K is the least famous with only 10 peers. In each experiment,10.000 distinct queries were issued starting at randomly select nodes that targeted a single existing resource.

4) Overall Performance Results: This section presents the overall results for queries in the scenario described above. Results were not discriminated by different categories as the goal of these results is to provide an overall overview of the system's performance. This section presents results only for $k = 256$, however the same tests were executed for $k = 128$ generating similar results.

As in Fig 2(a), *Curiata* outperforms the alternative architectures, both in terms of success and recall rates. Fig 2(b) presents latency values. Due to the DHT initial routing of queries *Curiata* has significant lower latency times. Notice that the lack of DHT support leads to a scenario where no significant differences can be found in the number of hops required to find the first hit for a query. Additionally, the self-organizing topology of the curia combined with the DHT routing allows *Curiata* to present a much lower



(a) Recall and Success Rate($k = 256$)



(b) Hop Count ($k = 256$)

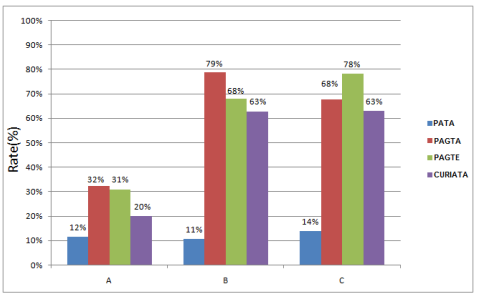
Figure 2. Overall Performance Results

number of hops required for locating the second and third hit of each query. Note that these results, only take into account the hop count of successful queries (this explains why, in Fig 2(b), the hop count for the third hit has lower values than the second hit hop count when using a $k = 256$ using the *Curiata* system). As expected, the performance of the evaluated solutions vary for queries that target items belonging to categories with different popularity (*i.e.*, with a variable number of nodes belonging to it). In order to evaluate the effects of resource category popularity, the following section provides detailed results for different scenarios.

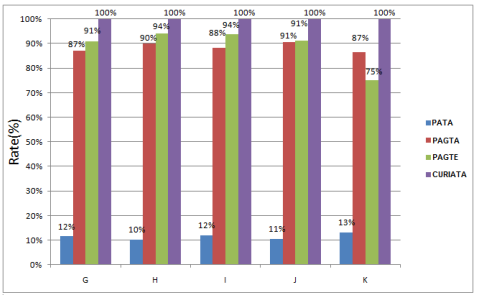
5) Performance According to Category: Fig 3 presents results for queries that target resources in rare categories (namely, categories G, H, I, J, and K) and common categories (categories A, B, and C). Due to space constrains only results for $k = 256$ were plotted, as this is the better scenario for the remaining considered architectures.

Comparative performance results show that *Curiata* brings no advantages when looking for resources in common categories. This happens because the DHT is unnecessary when the categories are very popular (given that resources from common categories are available in every region).

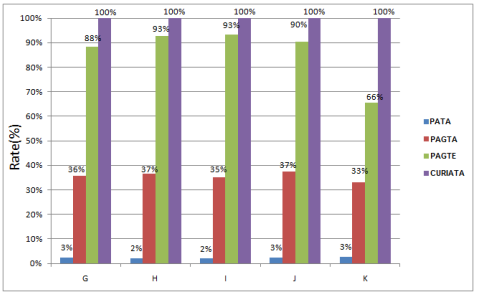
In sharp contrast, *Curiata* excels when searching resources from rare categories. For those resources, *Curiata* can achieve both a perfect recall and success rates and outperforms the remaining architectures in terms of latency. In the evaluated scenario, *Curiata* can locate every resource. Fig 3(d) depicts detailed results for latency only for the rarest categories J and K. Results show that *Curiata* offers a



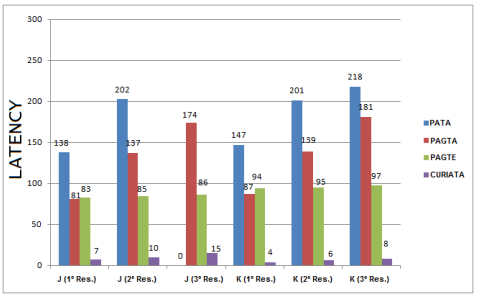
(a) Success Rate (common)



(b) Success Rate (rare)



(c) Recall Rate (rare)



(d) Hop Count (rare)

Figure 3. Performance Results for Rare and Common Categories

significant latency gain when compared to the other alternatives. This steams from the DHT routing that places queries in the relevant region of the self-organizing unstructured overlay. Since these regions are small, our system can easily visit all relevant nodes, effectively locating resources that are targeted by the query.

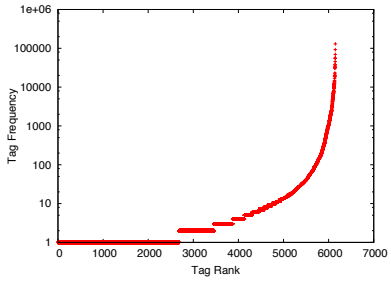


Figure 4. Tag Frequency for Resources

C. The Last.fm Dataset

In this evaluation scenario, data from 10.000 Last.fm user profiles and their top-50 most listened to tracks was retrieved. In the simulation architecture, the user's most listened tracks were their shared files/resources and the associated tags were used as categories in *Curiata*. In particular, the top 3 most voted tags of each track were used to categorize them.

Analysis of the retrieved data shows that for these 10.000 users there were 197.018 unique music tracks (from a total of 492.061 tracks), and these tracks were tagged by Last.fm users with no less than 6.150 different tags (considering only the 3 most voted tags of each track). In addition, the average number of tags per user was approximately 23.9 (note that this regards only the top-50 most listened to tracks by each user). Fig. 4 depicts the frequency of tags in all tracks. Out of curiosity, the most common tag is "Alternative" with 129.905 tagged tracks and the second one is "Indie" with 92.830 tracks.

This presents a challenging scenario for *Curiata*. Firstly, because there are some very popular tags/categories that are owned by a large portion of peers. This causes the search region for such categories to be very large and, therefore, a low TTL value might not be enough to find the desired resources. In addition, there are in average approximately only 2.5 copies of each resource in a universe of 10.000 users, which represents a very low replication factor of each file. Furthermore, the number of categories that each user belongs to is high (23.9 in average), especially considering only the top-50 most listened to tracks are being used. This high number of categories per peer makes it too costly for each peer to establish connections for all categories, causing some categories to be ignored during the biasing process. This may cause some resources to be unreachable since they do not belong to the top categories of their respective owners.

For the system simulation, the active view size was set to $d = 45$, peers biased their active views based on their top-15 tags/categories (the ones they had the most resources of), and can compete to become consuls for any of these 15 categories. The minimum number of neighbours for each

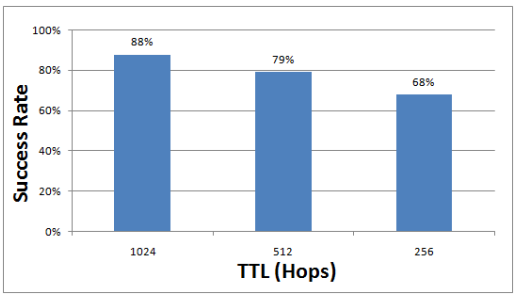


Figure 5. Success Rate for various TTL values

category was set to 3 ($smin = 3$) and the radius of the consul beacons was $r = 3$. This setup led around 1.400 peers to join the structured layer (DHT) in which 864 categories were represented. In addition, 1.063 resources (out of the 492.061) had none of their categories represented on the structured layer and therefore were unreachable using *Curiata*'s standard search protocol (although, they could still be reached using only a biased random walk). 10.000 queries were then issued originated in random nodes and targeting unique random resources. Different values of TTL were tested and thus obtained the results for the success rate (percentage of queries that found at least 1 matching resource) of such queries. These queries targeted only a single music track and a copy of the query was routed towards each category of the resource that was represented in the DHT. Each of these copies would be disseminated with a certain value of TTL (Fig. 5).

As expected, the success rate increases with the TTL value. This is mainly due to the existence of some very popular categories such as "Alternative" which 6.731 peers have in their top-15 categories or "Hip-Hop" which 4.699 peers have in their top-15 categories. Thus, the search region for these categories is huge and it is not easy to find copies of the target resources (note that the song tracks have in average 2.5 copies in the whole system) using these categories. However, when other (less popular) categories for the target resource are also represented in the structured layer, *Curiata* can use those those categories, which have smaller regions in the unstructured overlay, to find matches quicker. Therefore, even with a TTL of 256, the success rate is already close to 70%. To sum up, although *Curiata* did not achieve a perfect success rate, the results are encouraging, given the scenario used. More specifically, it is important to notice that resources have a very low replication rate (only 2.5 copies in average), and so this scenario strengthens the idea that *Curiata* can achieve a high hit-rate for rare resources.

V. Related Work

The simplest approach to perform resource location in P2P systems is to use a centralized scheme, where a single node is responsible for maintaining information about the location of all resources available in the system [1]. Since a central index maintains global knowledge of all resources available in the entire system, it can easily process complex queries. However, the overhead imposed on a single node, to maintain full and updated information concerning all resources, and to process queries on behalf of all peers, can be excessively high in a large-scale dynamic P2P environment.

Structured P2P systems implement distributed hash-tables, that support distributed exact-match lookups in a number of hops logarithmic with the system size [8]. Unfortunately, DHTs provide limited support for complex queries, as decomposing a complex query into a set of exact queries is often non-trivial and may even be impossible. Andrzejak and Xu[9] proposed Space Filling Curve over CAN construction to deal with range queries in DHTs. However, this approach is unable to provide full query flexibility, as the overlay has to be built taking into consideration a specific search space. The work by Reynolds and Vahdat [10], proposes a mechanism to add support for multiple search terms to DHT's. In order to achieve this goal, resources are inserted in the DHT using a set of associated keywords, such that each peer is responsible for maintaining the correspondence between keywords and respective resources. In order to process a query, a peer is required to obtain resource lists for each keyword and calculate their intersection. When compared with *Curiata*, this solution requires additional communication overhead, specially when searching for common keywords. Additionally, nodes in the DHT are required to maintain large lists of resources, which may be cumbersome for nodes responsible for common keywords.

Given the limitation of structured overlays to support non-exact queries, it becomes attractive to rely on unstructured P2P overlays, given that these overlays have a maintenance cost that is significantly lower. The most straightforward technique to implement resource location on top of unstructured overlays is through the use of flooding with limited horizon[11]. However, flooding is very expensive and, despite its large cost, may be ineffective when it comes to finding rare/unpopular resources that may exist in only a few peers (and, therefore, may not be located in the vicinity of the query source). Queries may also be propagated using random-walks[12], or guided-walks[13], which are less costly but exhibit a higher latency and have a lower recall. Query efficiency may be improved using techniques such as biasing the overlay network to approximate a small-world network, replicating all indexes in the one-hop neighborhood, and routing queries to high degree nodes. GIA[14] is a well known example of a system that combines these techniques, however such solutions not only have the overhead of maintaining additional state, as typically lead configurations where queries are only effectively processed by a small fraction of participants. Additionally, such solutions are not tailored to deal with queries that target rare resources.

As a way to avoid the cost of blind search, some unstruc-

tured systems use attenuated bloom filters as a strategy to embed information in the topology that can be used to bias the routing of random walks, increasing the probability of queries reaching relevant nodes. In some sense, our approach also use a similar strategy however, contrary to our approach in which a self-organizing strategy is employed to bias the topology of the overlay, solutions based on bloom filters cannot ensure the proximity of nodes that own similar resources. Therefore, the usefulness of attenuated bloom filters depends on the distribution of resources in the system whereas our approach strives to approximate nodes with similar content allowing queries to be routed with increased precision.

To overcome the scalability issues of unstructured systems, some systems propose the use of super-peers [15], such that nodes organize themselves in a two level hierarchy. Super-peers at the top level maintain consolidated indexes for the resources maintained by the group of regular peers that connect to them. Super-peers form an unstructured overlay of their own, which is used to disseminate queries. Unfortunately, in these systems super-peers process most of the queries. Additionally the maintenance costs of replicated indexes may easily become prohibitively high in face of system dynamics. Our approach also relies in an two level hierarchy topology. However, in *Curiata*, all participants actively contribute to the dissemination and processing of queries. Additionally, our approach does not require consolidated indexes to be mantained for neighbors, only generic information concerning the categories of their resources.

VI. Conclusions

In this thesis a new solution was proposed to support efficient search in large-scale peer-to-peer systems. Our solution combines the usage of systems based on both unstructured and structured overlay networks for a more efficient and flexible resource location solution. Our architecture was named *Curiata*, and aims at supporting flexible querying, like most unstructured solutions, while retaining the speed and efficiency provided by structured (DHT-based) solutions.

The architecture was applied to two different case-studies: a generic file sharing system and a resource discovery and allocation system for the cloud. The performance of the resulting systems has been extensively evaluated using simulations. The results have highlighted the benefits of our approach, namely: i) Quick and efficient resource location in rare categories which typically represents a challenging subject in P2P resource location systems; ii) Effectively approximates nodes that share resources belonging to the same categories and use this network organization to quickly find additional results; iii) Achieving high recall and success rates even when using reduced values of TTL in queries.

And also its limitations: i) The additional overhead introduced by the structured layer when searching for resources in very common categories; ii) Under scenarios with very high category heterogeneity, some objects may be unreachable through efficient routing.

Future plans include employing the Cubit[16] DHT in Curiata to develop a decentralized tracking and torrent search infrastructure that can cope with user errors when describing and tagging torrents and content.

Acknowledgments This work was partially supported by project "Redico" (PTDC/EIA/71752/2006) and by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. Parts of this work have been performed in collaboration with other members of the Distributed Systems Group at INESC-ID, namely, João Leitão and João Paiva.

References

[1] S. Flanning, "Napster," 1999, http://www.napster.com.

[2] D. Tsoumakos and N. Roussopoulos, "Analysis and comparison of p2p search methods," in *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*. New York, NY, USA: ACM, 2006, p. 25.

[3] B. Cohen, "Bittorrent," 2003.

[4] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "Seti@home-massively distributed computing for seti," *Computing in Science and Engineering*, vol. 3, no. 1, pp. 78–83, Jan/Feb 2001.

[5] J. C. A. Leitao, J. P. S. F. M. Marques, J. O. R. N. Pereira, and L. E. T. Rodrigues, "X-bot: A protocol for resilient optimization of unstructured overlays," in *SRDS'09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 236–245.

[6] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, June 2005.

[7] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," 2009, `http://peersim.sf.net`.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01*. New York, NY, USA: ACM, 2001, pp. 149–160.

[9] A. Andrzejak and Z. Xu, "Scalable, efficient range queries for grid information services," in *In Proc. of the 2nd P2P'02*. Washington, DC, USA: IEEE Comp. Society, 2002, p. 33.

[10] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in *(Unpublished Manuscript)*, 2002, pp. 21–40.

[11] D. Tsoumakos and N. Roussopoulos, "Analysis and comparison of p2p search methods," in *InfoScale '06*. New York, NY, USA: ACM, 2006, p. 25.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS' 02*. New York, NY, USA: ACM, 2002, pp. 84–95.

[13] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *ICDCS'02*, 2002, pp. 23–32.

[14] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM '03*. New York, NY, USA: ACM, 2003, pp. 407–418.

[15] B. Yang and H. Garcia-Molina, "Designing a super-peer network," *ICDE*, vol. 0, p. 49, 2003.

[16] B. Wong, A. Slivkins, and E. G. Sirer, "Approximate matching for p2p overlays with cubit," Computing and Information Science Technical Report, Cornell University, Tech. Rep., Dec. 2008.