

Efficient Discovery of Widely Distributed and Non-Volatile Resources on DHTs

Carlos Abalde, Víctor M. Gulías and Laura M. Castro
MADS Group. Computer Science Department
University of A Coruña. Spain
{cabalde, gulias, lcastro}@udc.es

Abstract—In the last years, a new generation of structured and decentralized P2P content distribution networks based on distributed hash tables (DHTs) has emerged. Nowadays, DHT algorithms are a powerful building block when designing distributed behaviors. However, sometimes DHTs are not flexible enough and problems arise when an efficient lookup in the DHT structure itself -instead of in the stored data- is required.

Our motivation scenario is the resource discovery problem in a decentralized and distributed caching architecture, deployed in a set of clusters built from cheap off-the-shelf computers organized in a DHT overlay network. In this paper we present the design of a resource discovery service layered on the underlying DHT and based on the combination of (1) an spanning-tree built mapping DHT nodes to their parents in a bottom-up fashion, and (2) a set of routing indices which allow nodes to efficiently lookup other nodes in the DHT overlay network, matching some resource constraints. This service does not alter the underlying DHT behavior, it scales to large wide area systems and it tracks both relatively static and frequently changing resources. Furthermore, the results of the experiments conducted to evaluate the good performance and scalability of the architecture are also presented.

Index Terms—Peer-to-Peer, Distributed Hash Table, Resource Discovery, Routing Index, Spanning-Tree.

I. INTRODUCTION

Much recent work on building scalable peer-to-peer (P2P) systems has focused on *distributed hash tables* (DHTs). DHTs offer a number of advantages over previous P2P systems (e.g., Napster, Gnutella, etc.), however, its lack of flexibility on efficient non key-based lookups is a well-known problem. Some recent research has addressed the problem [5], [11], [12], [21] suggesting new DHT algorithms or improvements to previous ones in order to allow a wider range of queries (e.g., keywords match, attribute ranges, etc.). Still, there is a functionality which can neither be addressed using a standard DHT, nor using a DHT extended with a wider range of supported queries: the lookup in the DHT structure itself, instead of the lookup in the data stored in the DHT.

The main contribution of this paper is the design and evaluation of a DHT resource discovery service based on the combination of (1) a *spanning-tree* built mapping DHT nodes to their parents in a bottom-up fashion, and (2) a set of *routing indices* which allow nodes to efficiently lookup other nodes in the DHT overlay network, matching some resource constraints.

The main advantages of our contribution are: (1) it does not alter the underlying DHT behavior; (2) it does not rely on centralized indexes nor on super-peers; (3) it scales to

large wide area systems; (4) it tracks both relatively static and frequently changing node resources; (5) it supplies a complete query language to express per-node resource constraints; and (6) it is flexible enough to adapt to a multi-level environment.

The rest of the paper is structured as follows. Section II states the problem, describing the system and its working environment. Related work to tackle this problem is discussed in section III. Then, our system data model, architecture and algorithms are presented in section IV. Section V shows some experimental results. Finally, in section VI, the contributions of this paper are summarized and the future work is described.

II. SYSTEM DESCRIPTION

Consider a distributed architecture with n heterogeneous nodes $\mathcal{N} = \{n_1, \dots, n_n\}$ organized in a DHT overlay network. Suppose that $\mathcal{R} = \{r_1, \dots, r_r\}$ are the names of different intra-node resources which combined define the current state of node $n_j \forall n_j \in \mathcal{N}$. So, $r_i \in \mathcal{R}$ could denote, for instance, the available node disk capacity, the available node network transfer bandwidth, the total node workload (CPU, size of pending work queue, etc.), the amount of free memory space, the number of networking interfaces, etc. The goal is to characterize the internal state of all DHT nodes at any time in order to efficiently find a DHT node matching some resource constraints. This can be a useful infrastructure to, for example, schedule a given task in a particular node.

Now, suppose that $r_i : n_j \rightarrow \mathbb{N} \forall r_i \in \mathcal{R}, n_j \in \mathcal{N}$ denotes a function which returns the current value of a resource r_i on node n_j . Note that, without loss of generality and for the sake of simplicity, resource values are represented as (or can be transformed in) integers. Therefore, the current state of node n_j , $\mathcal{S}(n_j)$, is modeled as a set of integer values,

$$\mathcal{S}(n_j) = \{r_1(n_j), r_2(n_j), \dots, r_r(n_j)\} \quad \forall n_j \in \mathcal{N}$$

Once the internal state of every DHT node is characterized as a set of resources, a scalable and simple resource querying infrastructure is required. Three different types of interesting queries have been identified:

- 1) *Aggregation queries*, where some node resources are aggregated. For example, a query asking for the total free disk capacity of the whole system.
- 2) *Filtering queries*, where a node or set of nodes verifying some filtering restriction over its resources is looked for. Two interesting subtypes have been identified:

- a) *Single-attribute range queries*, where the restriction is expressed as a single range condition. For example, a query asking for a couple of nodes with at least one available unit of network transfer bandwidth.
 - b) *Multi-attribute range queries*, which are a conjunction of the previous type of queries.
- 3) *Selection queries*, where a node or set of nodes verifying some group property over its resources is looked for. For example, a query asking for the set of five nodes with the largest number of available units of network transfer bandwidth. Note that selection and filtering queries can be combined using logical operators.

This paper focuses on these types of queries, especially on filtering queries, implementing disjunction of queries by multiple distinct requests. Additionally, the proposed solution does not introduce special system constraints or modifications to the underlying DHT behavior, and does not rely on centralized indexes nor on super-peers.

Some important assumptions about the DHT environment have been made here. The nodes are connected in a private wide area network, and their arrival, failure and departure rates are under the normal limits in a controlled environment (i.e., churn is not a realistic situation). In other words, the system deals with a set of widely distributed resources that are not volatile. These assumptions are acceptable since the motivation and working environment where the presented approach is applied is a decentralized, distributed and scalable caching architecture deployed in a set of clusters built from cheap off-the-shelf components [10].

III. RELATED WORK

Resource discovery, reservation, scheduling and monitoring are key elements in Grid architectures. From the predominant MDS Globus monitoring and discovery system [1] to the variety of P2P models for resource discovery on Grids reported by Trunfio et al [19], a full range of P2P approaches have been identified.

Van Renesse et al propose Astrolabe [17], based on an unstructured Grid gossip protocol with a high degree of replication, especially adapted for read-dominated attributes. Talia et al [18] present a framework for resource discovery in Grids where each resource is mapped to a DHT, which is used for exact and range queries over static resources. An additional DHT is created and used as the underlying flooding infrastructure for arbitrary queries and queries over dynamic resources.

Not necessarily related to Grid environments, SWORD [15] locates a set of machines matching user-specified constraints on both static and dynamic node attributes. A DHT is used like an inverted index where each attribute is mapped to a specific DHT subzone. An expressive XML-based query language, where both single-node and inter-node characteristics can be constrained, is provided, and penalty functions to fine-tune queries are also introduced. SWORD, being completely

different to our approach to the problem, is an interesting option which will demand a further and deeper analysis.

Many systems like Cone [4], SOMO [23], Willow [20], DASIS [2] and SDIMS [22] are based on some kind of aggregation protocol. Cone augments a DHT with a a lightweight tree -one per-attribute and per-aggregation operation-. It supports aggregation and range queries over those resources, even though doesn't specify how to deal with multi-attribute queries. Rather than augmenting it, SOMO layers over a DHT and defines the concept of data overlay, which is a mechanism to implement arbitrary data structures on top of a DHT. The data overlay is used by SOMO to manage system resources, but it does not support range queries. DASIS uses a Kademia-like tree structure -although it argues that can be adapted to other topologies-. It aggregates information about DHT nodes which is used in the join algorithm to balance the P2P topology better than is typically achieved through random placement. Willow aggregates information in a quite similar way to DASIS, and last but not least, SDIMS extends the Pastry DHT, mapping each attribute and aggregation operation to a tree and providing an API that lets applications control the propagation of read and writes based on resource characteristics.

Finally, Marzolla et al [14] propose a couple of P2P systems for resource discovery based on routing indexes [6]. Each Grid node maintains, for each attribute that characterizes the managed resources, a bitmap index describing the local presence/absence of those resources. Those bitmaps are managed and aggregated using an *ad-hoc* tree-structured overlay network where system nodes are organized. These systems, to the best of our knowledge, are the closer approaches to our contribution. However, Marzolla et al argue that P2P networks, when less structured than DHTs, are more suitable to deal with dynamic data. Therefore, their resource discovery approach is based in specific P2P topologies (a tree vector and a forest of trees). Our approach, on the contrary, is deliberately based on a DHT whose nodes are organized in a spanning-tree. Furthermore, routing indexes are also used at every tree level, but they aggregate a more detailed and still compact description of the state of the system resources.

IV. SYSTEM DATA MODEL AND ARCHITECTURE

Any DHT node can receive a query asking for a set of DHT nodes matching some resource constraints. The node receiving the query first evaluates it against its own internal state, and, if itself matches the constraints, reports its identity to the client node, which evaluates a stop condition and returns the result to the reporting node. If the stop condition has not been reached, the node selects a set of its neighbors to forward the query to (along with some state information). In turn, each of the neighbors evaluates the query in a similar fashion, returns the results to the requesting client, and forwards the query to other neighbors until the stop condition is reached.

As shown in figure 1, all DHT nodes are dynamically configured in a bidirectional tree structure which is used to define the neighborhood relationship. However, although many

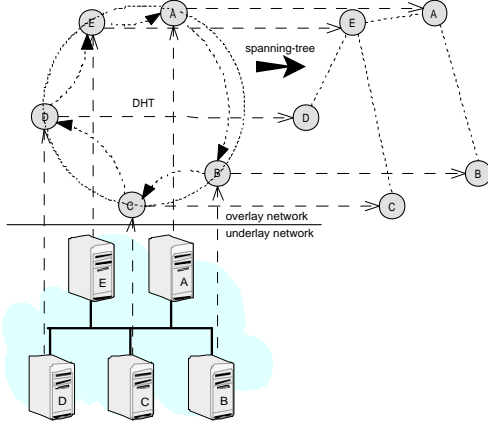


Fig. 1. DHT spanning-tree mapping

systems exporting a simple and general DHT interface are internally, directly or indirectly, based on tree structures where DHT nodes are aggregated and where lookup requests are routed, since DHT independence is a key requirement in this system, first of all, a decentralized, flexible and scalable DHT tree construction algorithm layered over the DHT is required. This problem is closely connected with the spanning-tree definition in application-level broadcasting and multicasting systems over DHTs. There is a large body of literature which addresses that problem. In general, existing schemes can be categorized either in top-down/bottom-up spanning-tree approaches [7], [13] or in flooding-based approaches [16]. The major drawback of the latter is the redundant traffic generated, while the major drawback of the former is that operation under churn or node failures on the middle of the hierarchy results in temporal disconnections from some tree-DHT-zones.

In this case, since the DHT environment is assumed to be a mainly-stabilized scenario, the main drawback of the top-down/bottom-up approaches is negligible. Therefore, the bottom-up approach proposed by Li et al [13] has been used here as the underlying tree-building infrastructure because of its simplicity, clarity and good experimental results. The following section sums up the DHT mapping process using this technique.

A. Bottom-up Spanning-Tree Construction

Li et al present in [13] a scalable and robust algorithm to build efficient aggregation/broadcast trees over DHTs. The key idea of the tree-building algorithm is to use a many-to-one function, $P(x)$, to map each ring-based DHT node uniquely to a parent node based on its identifier. More specifically, the parent node for node $n_j \in \mathcal{N}$ is the node responsible for key $P(n_j)$. The parent function $P(x)$ satisfies the following conditions,

$$\begin{aligned} P(\alpha) &= \alpha \\ P^\infty(n_j) &= \alpha, \forall n_j \in \mathcal{N} \\ D(P^{m+1}(n_j), \alpha) &\leq D(P^m(n_j), \alpha), \forall m > 0; n_j \in \mathcal{N} \end{aligned}$$

where α is the identifier of the root of the tree, n_j is any valid node identifier and $D(x, y)$ is a distance metric defined for two valid node identifiers x and y in the DHT namespace. There are many functions that satisfy those conditions. The following is a simple example,

$$P(x) = \begin{cases} \lfloor \frac{x}{k} \rfloor \bmod m & \text{for } 0 \leq x < \frac{m}{2} \\ \lceil m - \frac{m-x}{k} \rceil \bmod m & \text{for } \frac{m}{2} \leq x < m \end{cases} \quad (1)$$

where x is a node identifier, k is a parameter that determines the branching factor of the tree, and $m = 2^{\text{address space bits}}$.

Then, given $P(x)$ and a node n_j , the function $\mathcal{P}(n_j)$ to build the spanning-tree \mathcal{T} over \mathcal{N} is defined. Let be Chord the underlying DHT algorithm and $\text{Succ}(key)$ the operation which returns the successor node -responsible node- for key key . Therefore, $\mathcal{P}(n_j)$ is defined as,

$$\mathcal{P}(n_j) = \begin{cases} n_j & \text{if } P(n_j) = \alpha \\ n_k & \text{if } \begin{cases} \text{Succ}(P^{m-1}(n_j)) = n_j & \forall m > 1 \wedge \\ \text{Succ}(P^m(n_j)) = n_k & k \neq j \end{cases} \end{cases}$$

B. Routing Indices Definition

The tree structure is used as the foundation to explore the DHT node space. Since any node in the DHT can be asked for a node or set of nodes matching some resource constraint, a distributed heuristic function is defined in order to direct the search process in the spanning-tree and minimize the number of required hops.

For that, the routing indices (RIs) and compound routing indices (CRIs) proposed by Crespo et al [6] have been reinterpreted and adapted to this problem. As well as the different use of RIs and CRIs (containers lookup, instead of contents lookup), all the complexity related with the management of cycles in the P2P network is avoided (RIs are devised in the context of an unstructured P2P network).

In order to achieve the goal, let $\mathcal{RJ}(r_i, n_j)$ be the routing index of resource r_i at node n_j . Moreover, let $\mathcal{T}(n_j)$ be the subspanning-tree of \mathcal{T} where $n_j \in \mathcal{T}$ is the root node. Thus, $\mathcal{RJ}(r_i, n_j)$ is the aggregated representation of the current state of the resource r_i for all nodes in $\mathcal{T}(n_j)$. For every child node n_x of n_j ($\mathcal{P}(n_x) = n_j$), $\mathcal{RJ}(r_i, n_j)$ aggregates the state of the subtree $\mathcal{T}(n_x)$, allowing the definition of a heuristic function to direct the lookup process,

$$\mathcal{RJ}(r_i, n_j) = \{(n_x, \mathcal{CRJ}(r_i, n_x))\} \quad \forall n_x / \mathcal{P}(n_x) = n_j$$

where $\mathcal{CRJ}(r_i, n_x)$ is the CRI of the subspanning-tree $\mathcal{T}(n_x)$,

$$\mathcal{CRJ}(r_i, n_x) = \{\mathcal{CRJ}(r_i, n_y) \mid \forall n_y / \mathcal{P}(n_y) = n_x\} \otimes \mathcal{S}(r_i, n_x)$$

$$\mathcal{S}(r_i, n_j) = \{(1, r_i(n_j), r_i(n_j))\}$$

where \otimes is the compound operator whose behavior depends on r_i (see section IV-E).

Figure 2 shows the calculation process of $\mathcal{RJ}(r_1, n_j)$. Let r_1 represent the available node disk capacity. Consider that n_y has 10 GB available ($\mathcal{S}(r_1, n_y)$), n_z has 25 GB available ($\mathcal{S}(r_1, n_z)$) and n_x has 10 GB available ($\mathcal{S}(r_1, n_x)$). Since both n_y and n_z are leaf nodes in \mathcal{T} , their corresponding

$\mathcal{S}(r_1, n_y) = \{(1, 10, 10)\}$ $\mathcal{RJ}(r_1, n_y) = \emptyset$ $\mathcal{CRJ}(r_1, n_y) = \mathcal{S}(r_1, n_y)$	$\mathcal{S}(r_1, n_z) = \{(1, 25, 25)\}$ $\mathcal{RJ}(r_1, n_z) = \emptyset$ $\mathcal{CRJ}(r_1, n_z) = \mathcal{S}(r_1, n_z)$
$\mathcal{S}(r_1, n_x) = \{(1, 15, 15)\}$ $\mathcal{RJ}(r_1, n_x) = \{(n_y, \{(1, 10, 10)\}), (n_z, \{(1, 25, 25)\})\}$ $\mathcal{CRJ}(r_1, n_x) = \mathcal{S}(r_1, n_x) \otimes \mathcal{CRJ}(r_1, n_y) \otimes \mathcal{CRJ}(r_1, n_z) = \{(2, 10, 15), (1, 25, 25)\}$	

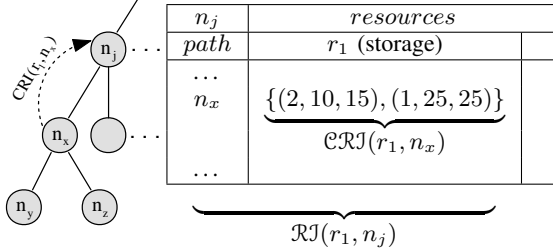


Fig. 2. Routing index at node n_j

RIs $\mathcal{RJ}(r_1, n_y)$ and $\mathcal{RJ}(r_1, n_z)$ are empty sets. Their CRIs, notified to n_x , simply aggregate the local state. Combining this information, n_x creates its own RI $\mathcal{RJ}(r_1, n_x)$, which is also aggregated with its local state using the \otimes operator to calculate its own CRI $\mathcal{CRJ}(r_1, n_x)$, which is notified to the parent node n_j . Figure 2 also shows the table data structure storing the RI at node n_j . Note that, in the example, a compound operator \otimes that constraints to two the number of subintervals in a CRI has been used.

Regardless of the compound operator used, $\mathcal{RJ}(r_i, n_j)$ is composed by a set of tuples $(path, \{(hits, lbound, ubound)\})$, where $hits \in \mathbb{N}$ is the number of DHT nodes $n_k \in \mathcal{T}(path)$ ($\mathcal{P}(path) = n_j$) where $r_i(n_k) \in [lbound, ubound]$. Therefore, recalling example shown above, using its local RI, node n_j knows that following its child n_x there are three different DHT nodes, two whose available disk capacity is in the interval $[10, 15]$ and one whose its available capacity is exactly 25.

C. RIs Creation and Maintenance

Once a node n_j joins the DHT and it is mapped to the spanning-tree by the bottom-up spanning-tree construction algorithm, it initializes its local RI (RI array) and CRI (CRI array) as shown in algorithm 1. In the algorithm comments are written to the right of $//$ to ease reading and the notation $n_j.foo()$ stands for the function $foo()$ being invoked and executed at node n_j .

Algorithm 1 $n_j.init()$

```

for all  $r_i \in \mathcal{R}$  do
   $RI[r_i] = \emptyset // \mathcal{RJ}(r_i, n_j)$ 
   $CRI[r_i] = \emptyset // \mathcal{CRJ}(r_i, n_j)$ 
end for
  stabilize()

```

Since n_j owns a new set of resources, it also starts a stabilization cycle in order to made them available to the whole system. The stabilization pseudocode is shown in algorithm 2.

The CRI is calculated aggregating the local resources and the local RI using the compound operator. If n_j notices a relevant change between the previous notified CRI and the current one, the parent node in the spanning-tree is notified invoking its *update* procedure.

Algorithm 2 $n_j.stabilize()$

```

 $CRI_{old} = CRI$ 
for all  $r_i \in \mathcal{R}$  do
   $S = \{(1, r_i(), r_i())\} // \mathcal{S}(r_i, n_j)$ 
   $CRI[r_i] = \otimes(S, RI[r_i][n_{x_1}], \dots, RI[r_i][n_{x_k}])$ 
end for
if  $\mathcal{P}(n_j) \neq n_j \wedge CRI \neq CRI_{old}$  then
   $\mathcal{P}(n_j).update(n_j, CRI)$ 
end if

```

Algorithm 3 describes the actions taken by a node n_j when an update request from a child node n_x is received. It updates its local RI with the new information and starts a new stabilization cycle in order to spread the modifications in the spanning-tree hierarchy. Note that outgoing notifications could be batched together for efficiency and could also be delayed based on distance measurements between the last notified and current CRIs.

Algorithm 3 $n_j.update(n_x, CRI_x)$

```

for all  $r_i \in \mathcal{R}$  do
   $RI[r_i][n_x] = CRI_x[r_i]$ 
end for
  stabilize()

```

The stabilization cycles are run by every DHT node periodically in order to keep all the routing indices up-to-date, and therefore to guarantee the correctness of resource lookups. Besides those regular stabilizations, additional stabilization cycles must be triggered on events like changes on the spanning-tree hierarchy -parent change-, changes on local resources availability and so on.

In contrast to the flooding strategy proposed in [14] to publish changes, our system is based on a bottom-up approach. This approach allows updates to be pagated in $O(h)$ steps, where h ($h \ll n = |\mathcal{N}|$) is the average height of \mathcal{T} , which is a value dependent on $P(x)$.

This approach is efficient, allows the management of both static and dynamic node resources, and is compatible with all the identified types of queries. A bottom-up-down strategy would improve the system behavior dealing with selection queries. However, it would also complicate the algorithms and, depending on the update implementation, it would require up to $O(n)$ steps. In any case, our approach is compatible with the SDIMS [22] approach, where each resource can be parameterized with a different spreading behavior.

D. RIs Usage

Based on RIs, any DHT node $n_j \in \mathcal{T}$ must select a subset of its most suitable children to forward a query to, or, if

the resulting set is empty, forward the query to the parent node. Therefore, given a query, the goodness (i.e., the value of the heuristic function) of each child for that query need to be computed by n_j . To do that, the number of nodes in $\mathcal{T}(n_j)$ matching the query must be estimated for every possible exploration route (i.e., every child of n_j). In [8], [9] some discussion about the different types of possible estimators is presented. Since these estimators are not the goal of this work, a simplified model where all $r_i \in \mathcal{R}$ are independent from each other and where every query is expressed as a conjunction of restrictions is used. Note that disjunction of queries is implemented by multiple distinct queries.

Therefore, if n_j is the node receiving a query Q like $\alpha \wedge \beta \wedge \gamma$, restrictions over whichever resources r_α , r_β and r_γ respectively, the value of the heuristic function H is computed for every child n_x . That value will be the estimation of the number of nodes in $\mathcal{T}(n_x)$ matching Q ,

$$H(Q, n_j \rightsquigarrow n_x) = |\mathcal{T}(n_x)| \times \frac{\mathcal{R}\mathcal{J}(r_\alpha, n_j \rightsquigarrow n_x)}{|\mathcal{T}(n_x)|} \times \frac{\mathcal{R}\mathcal{J}(r_\beta, n_j \rightsquigarrow n_x)}{|\mathcal{T}(n_x)|} \times \frac{\mathcal{R}\mathcal{J}(r_\gamma, n_j \rightsquigarrow n_x)}{|\mathcal{T}(n_x)|}$$

where n_x is one of the possible exploration paths (i.e., one of the children of n_j), $|\mathcal{T}(n_x)|$ is the number of nodes in $\mathcal{T}(n_x)$, and $\mathcal{R}\mathcal{J}(r_\alpha, n_j \rightsquigarrow n_x)$ is, according to the CRI $\mathcal{C}\mathcal{R}\mathcal{J}(r_\alpha, n_x)$, the number of nodes matching α . Definitely, node n_j estimates the success probabilities p_α , p_β , etc. following path n_x , and then combines them according to the conjunction operator which connects all the filtering restrictions as $\prod_{i=\alpha, \beta, \dots} p_i$. Obviously, a drawback of this approach is that it does not take into account the cost associated with the number of hops required to find a node in \mathcal{T} matching Q , even though some improvements proposed in [6] could be applied to our approach.

Algorithm 4 shows the lookup procedure pseudocode. A *client* $\in \mathcal{N}$ starts the process asking for a set of nodes matching a query Q . This event starts a directed depth-first search where the heuristic function H optimizes the process in order to minimize the number of hops. The client is notified with partial results until it decides that it has enough elements in the result set. If the result set required by clients is of size one, our approach allows locating results in $O(h)$ steps.

E. The Compound Operator

Every resource type $r_i \in \mathcal{R}$ has an associated compound operator $\otimes_{r_i} : \{(h, lb, ub)\} \longrightarrow \{(h, lb, ub)\}$ and a granularity $|r_i| \in \mathbb{N}$. Operator \otimes_{r_i} (from now on \otimes) gets a set of tuples like (h, lb, ub) (meaning “ $h \in \mathbb{N}$ hits in interval $[lb, ub]$ ”) and aggregates them in another set which cardinality is not greater than $|r_i|$.

As shown in section V, the precision of the lookup algorithm is directly affected by the set up granularities, which, at the same time, have a direct influence in the behavior of the \otimes operator. Granularities must be fine-tuned in order to establish a commitment between the lookup precision (greater granularities, better precision) and the storage and management costs of the RIs and CRIs (greater granularities, larger RIs and CRIs).

Algorithm 4 $n_j.lookup(client, Q)$

```

done = false
if  $n_j$  matches  $Q$  then
  done = client.report( $n_j$ )
end if
if done = false then
  rank =  $\emptyset$ 
  for all  $n_x \in RI[\cdot]$  do
    if  $H(Q, n_j \rightsquigarrow n_x) \geq 1$  then
      rank.append( $(H(Q, n_j \rightsquigarrow n_x), n_x)$ )
    end if
  end for
  rank.sort() // sort by first attribute
  for all  $(H, n_x) \in rank$  do
    done =  $n_x.lookup(client, Q)$  // if  $n_x$  is not caller
    if done then
      return true
    end if
  end for
  done =  $\mathcal{P}(n_j).lookup(client, Q)$  // if  $\mathcal{P}(n_j)$  is not caller
end if
return done

```

Regarding to the behavior of the \otimes operator, a generic implementation suitable for the most general cases has been used in this work. Let the distance between two intervals $[lb1, ub1]$ and $[lb2, ub2]$ be measured as $D([lb1, ub1], [lb2, ub2]) = D(lb2, lb1) + D(ub1, ub2)$, where $D(a, b) = b - a$ if $a < b$, 0 otherwise. Then, algorithm 5 summarizes the behavior of \otimes ,

Algorithm 5 $\otimes_{r_i}(I)$

```

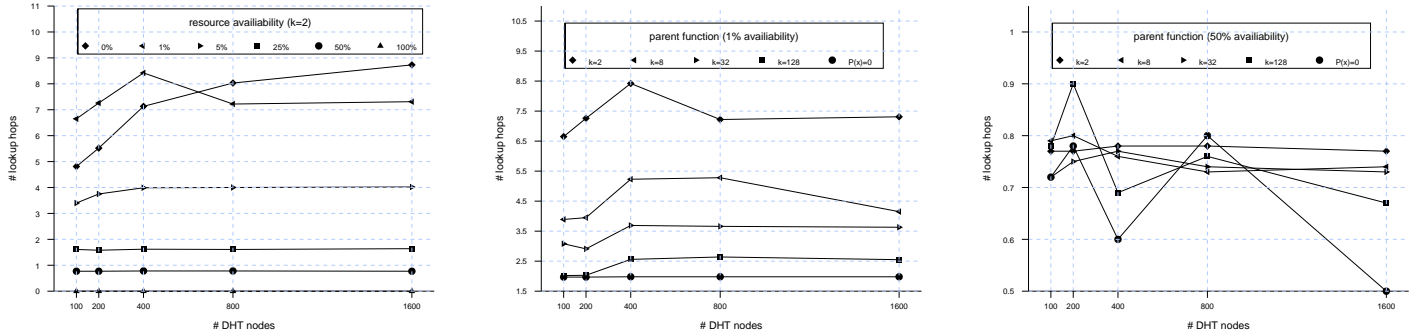
sort  $I$  from shortest to largest interval lengths
 $R =$  first  $|r_i|$  elements of  $I$ 
for all remaining intervals  $(h, lb, ub) \in I$  do
  select  $(h_i, lb_i, ub_i) \in R$  minimizing  $D([lb, ub], [lb_i, ub_i])$ 
   $R.remove((h_i, lb_i, ub_i))$ 
   $R.insert((h + h_i, min(lb, lb_i), max(ub, ub_i)))$ 
end for
return  $R$ 

```

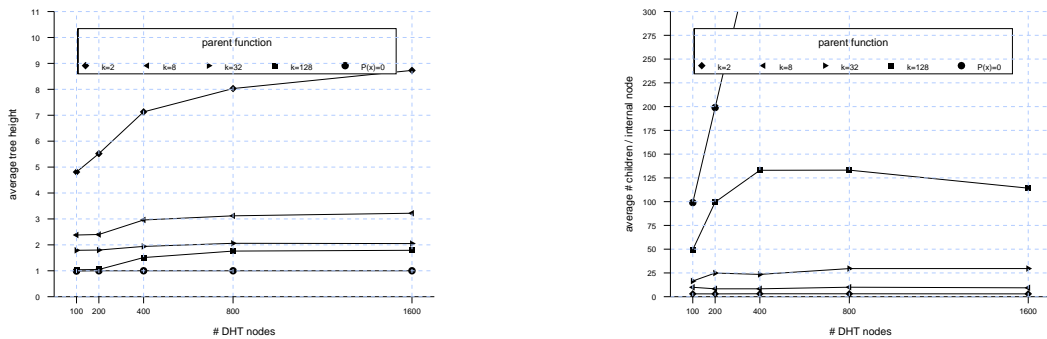
Note that, depending on the type of resource, it could be better to use user-defined compound operators. They could aggregate intervals following probability distributions, appropriately manage atypical values or even dynamically adjust the value of the granularity $|r_i|$.

V. EXPERIMENTAL RESULTS

A process-oriented simulation was developed using the concurrent functional language Erlang/OTP [3] and picking Chord as the algorithm for the DHT tier. First of all, the average number of hops of the lookup algorithm 4 when routing a single-attribute exact-match query $Q = 1 \leq r_1 \leq 1$ was measured. In the experiment, we considered a stabilized Chord DHT \mathcal{N} with a variable number of peers, ranging from



(a) Parent function shown in the equation 1 ($k = 2$) (b) 99% of $n_j \in \mathcal{N}$ $r_1(n_j) = 0$; 1% $r_1(n_j) = 1$ (c) 50% of $n_j \in \mathcal{N}$ $r_1(n_j) = 0$; 50% $r_1(n_j) = 1$
 Fig. 3. Number of hops routing a single-attribute exact-match query ($Q = 1 \leq r_1 \leq 1$; $r_1(n_j) \in [0, 1] \forall n_j \in \mathcal{N}$; $|r_1| = 2$)



(a) Average height (b) Average number of children per internal node

Fig. 4. Topological measurements of the spanning-tree \mathcal{T}

100 to 1600, a single resource r_1 which state could be 0 or 1, and a generic \otimes operator with $|r_1| = 2$. Figure 3 shows the results for three different scenarios. The data points were calculated by performing 20 iterations. For each iteration, a new uniformly randomized state of resource r_1 was built according with a global availability profile, ranging from 0% (r_1 exhausted) to 100% ($r_1 = 1 \forall node \in \mathcal{N}$). Then, every node in the DHT was instructed to route Q , measuring the average number of required steps.

Figure 3a shows the behavior of the system when $P(x)$ is defined according to equation 1 with a branching factor $k = 2$. As we can see, the average number of routing hops is not significantly affected by the network size, with an upper limit constrained by the average height of the spanning-tree $\mathcal{T} - O(h)$ - (see figure 4a). On the other hand, figure 3a also shows how the number of routing hops is affected by global resource availability (more availability, less routing hops).

Figures 3b and 3c show the behavior of the system in two different scenarios. The former, when r_1 is nearly exhausted, and the later, when r_1 is available in half of the DHT nodes. Again, as we can see, the average number of routing steps is independent from the network size and is constrained by h . The number of hops is only affected by the parent function $P(x)$ used to build \mathcal{T} , especially when there are only a few

candidate peers in the network (i.e., low resource availability). As we can see, a greater branching factor k reduces the number of hops. However, as shown in figure 4b, increasing k too much turns the system in a mainly-centralized approach (or fully centralized if $P(x) = 0$, with a $O(2)$ lookup steps) where a few internal nodes must route every lookup. Therefore, a careful design of the system must be performed in order to avoid this situation and to select an optimal parent function.

Secondly, an experiment to measure the impact of the compound operator granularity in the lookup process was conducted. We used the parent function shown in equation 1 with $k = 2$. We considered the same network \mathcal{N} and resource r_1 as in the previous experiment. However, in each iteration of this experiment r_1 could take any random and uniform value between 0 and 10000, and Q was defined as a single-attribute exact-match query looking for an existing value of r_1 .

Figure 5 shows the impact of $|r_1|$, ranging from 2 to 256, in the lookup precision. As noted in section IV-E, a careful design of the resource granularity based on the domain of each resource and on the most common type of queries, must be done in order to do not affect the precision of the lookup.

Finally, although a controlled environment is assumed, some minimal amount of node failures, arrivals and departures will be present in the real system. Due to space limitations, simula-

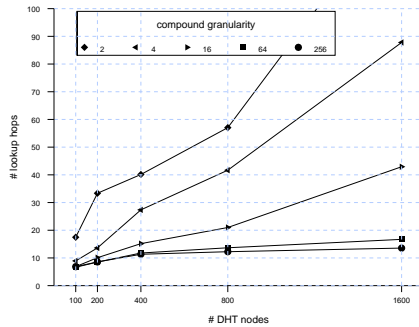


Fig. 5. Lookup precision routing a single exact-match query ($Q = V \leq r_1 \leq V$; $r_1(n_j) \in [0, 10000] \forall n_j \in \mathcal{N}$; $P(x)$ shown in equation 1; $k = 2$)

tions under these conditions are not presented, however, since the main routing structures (the DHT ring and the spanning-tree) are dynamic and based on soft-state information, the impact on the system behaviour is minimal.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a decentralized resource discovery service layered on the top of a DHT overlay network. Its design is especially oriented to a DHT-based caching architecture whose components are deployed in a private wide area network where churn is not a realistic scenario. Our service does not alter the underlying DHT behavior, does not rely on centralized indexes, scales to large wide area systems, tracks both relatively static and frequently changing node resources, supplies a complete query language to express per-node resource constraints, and it is flexible enough to adapt to a multi-level environment.

The conducted simulations have shown the good performance and scalability of the architecture, and the importance of doing a careful analysis of the environment to set up the service with an optimal parent function $P(x)$, stabilization rates, compound operators and resource granularity values.

As future work, our next steps involve (1) studying the behaviour of the system with more complex parent functions and routing indices approaches, and (2) carrying on with the design of the distributed caching architecture based on this resource discovery service.

ACKNOWLEDGMENT

This research has been partially supported by MEC Project TIN2005-08986.

REFERENCES

- [1] Globus Toolkit MDS: Monitoring & Discovery System. <http://www.globus.org/toolkit/mds/>, 2007.
- [2] Keno Albrecht, Ruedi Arnold, Michael Gahwiler, and Roger Wattenhofer. Aggregating information in peer-to-peer systems for improved join and leave. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, pages 227–234, Washington, DC, USA, 2004. IEEE Computer Society.
- [3] Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, July 2007.
- [4] R. Bhagwan, G. Varghese, and G. Voelker. Cone: Augmenting dhts to support distributed resource discovery. Technical Report CS2003-0755, University of California, 2003.

- [5] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the SIGCOMM Symposium on Communications Architectures and Protocols*, Portland, OR, August 2004.
- [6] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA, 2002. IEEE Computer Society.
- [7] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient broadcast in structured p2p networks. In *Proc. of the 2nd International Workshop On Peer-To-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, February 2003.
- [8] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. Precision and recall of gloss estimators for database discovery. In *PDIS '94: Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, pages 103–106, Washington, DC, USA, 1994. IEEE Computer Society.
- [9] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of gloss for the text database discovery problem. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 126–137, New York, NY, USA, 1994. ACM Press.
- [10] V. M. Gulías, M. Barreiro, and J. L. Freire. VoDKA: Developing a video-on-demand server using distributed functional programming. *Journal of Functional Programming*, 15(4), 2005.
- [11] Clément Jamard, Georges Gardarin, and Laurent Yeh. Indexing textual xml in p2p networks using distributed bloom filters. In *DASFAA*, pages 1007–1012, 2007.
- [12] Yuh-Jzer Joung, Chien-Tse Fang, and Li-Wei Yang. Keyword search in dht-based peer-to-peer networks. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 339–348, Washington, DC, USA, 2005. IEEE Computer Society.
- [13] Ji Li, Karen Sollins, and Dah-Yoh Lim. Implementing aggregation/broadcast over distributed hash tables. *SIGCOMM Comput. Commun. Rev.*, 35(1):81–92, 2005.
- [14] Moreno Marzolla, Matteo Mordacchini, and Salvatore Orlando. Peer-to-peer systems for discovering resources in a dynamic grid. *Parallel Computing*, 33(4-5):339–358, 2007.
- [15] David Oppenheimer, Jeannie Albrecht, David Patterson, and Amin Vahdat. Scalable wide-area resource discovery. Technical Report UCB/CS-D-04-1334, EECS Department, University of California, Berkeley, Jul 2004.
- [16] M. Portmann and A. Seneviratne. Cost-effective broadcast for fully decentralized peer-to-peer networks. *Computer Communications*, 26(11):1159–1167, 2003.
- [17] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.
- [18] Domenico Talia, Paolo Trunfio, Jingdi Zeng, and Mikael Höggqvist. A dht-based peer-to-peer framework for resource discovery in grids. Technical Report TR-0048, Institute on System Architecture, CoreGRID - Network of Excellence, June 2006.
- [19] P. Trunfio, D. Talia, H. Papadakis, P. Fragogioulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Gener. Comput. Syst.*, 23(7):864–878, 2007.
- [20] Robbert van Renesse and Adrian Bozdog. Willow: Dht, aggregation, and publish/subscribe in one protocol. *LNCS*, 3279:173–183, 2005.
- [21] Maria-Del-Pilar Villamil, Claudia Roncancio, and Cyril Labbe. Pins: Peer-to-peer interrogation and indexing system. In *IDEAS'04: Proceedings of the International Database Engineering and Applications Symposium (IDEAS'04)*, pages 236–245, Washington, DC, USA, 2004. IEEE Computer Society.
- [22] Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390, New York, NY, USA, 2004. ACM Press.
- [23] Z. Zhang, S. Shi, and J. Zhu. SOMO: Self-organized metadata overlay for resource management in P2P DHT. *LNCS*, 2735:170–182, 2003.