

# Adaptive routing in structured peer-to-peer overlays

Spyros Kotoulas and Ronny Siebes  
Vrije Universiteit Amsterdam  
Department of Computer Science  
De Boelelaan 1081, 1081HV, Amsterdam  
The Netherlands  
{kot,ronny}@few.vu.nl

## Abstract

*The OpenKnowledge project aims at knowledge sharing through open and flexible peer interactions. Within this project, we are developing a system that supports searching, developing and sharing of interactions/workflows consisting of roles implemented by software that can be shared and executed by peers. Its main requirements are openness, scalability, decentralization and robustness. Part of this system is a discovery service, which will be the focus of this paper. This service aspires to fulfill the above requirements featuring a Peer-to-Peer architecture and Distributed Hash Tables (DHTs) to achieve robustness through redundancy and scalability through decentralization. Resources are discovered using a set of attribute-value pairs. A straightforward DHT-based approach that creates a distributed inverted index suffers from a linear increase of messages and replicas with the number of attributes. We try to reduce this number by proposing an efficient multi-attribute routing algorithm. We emulate and test our implementation on the DAS-2 distributed supercomputer.*

## 1 Introduction

Peer-to-Peer is a promising technology addressing some of the major challenges in modern distributed systems since it provides scalability through distribution of the deployment cost and all peers have the same functionality, providing robustness by redundancy.

The EU-funded OpenKnowledge project<sup>1</sup> has as one of its goals to build a P2P system, which we call the OK-system, for sharing knowledge, not only in the form of data but also in the way the data is processed<sup>2</sup>. Using this system, people can publish workflow descriptions (also called *Interaction Models* or *IMs*), and peers can subscribe them-

selves to play one or more of the roles in them. The role-code (i.e. software) that a peer needs to have to play such a role can also be shared and downloaded via the OK-system. Peers can also subscribe themselves to be coordinators of IMs, being responsible for their correct execution. All components of the OK-system will be implemented in a way that everything runs distributed without any central control. In this paper we focus on the component responsible for finding:

- *Interaction Models* Interaction models define the way services interact, expressed in a formal language like LCC[11] or BPEL<sup>3</sup>. They are described by a set of attribute-value pairs, they have small size and their search is facilitated by multi-attribute search. In table 1 an example descriptor of an IM is given. The unique identifier of the IM is `Auction5443FF`, and it is described by a set of attributes pertaining to its use and characteristics. These attributes (e.g. `Type` or `Role`) are fixed for OpenKnowledge, although we will see later that our discovery service does not depend on such a fixed schema.
- *Service descriptions* Services are described by a (potentially large) set of attribute-value pairs. They are small in size, and expected to be transferred over the network often. Consequently, we need efficient mechanisms for multi-attribute search, incorporating methods from information retrieval. In table 1 we show an example descriptor of a service description. For example, a user that has found the example interaction model can search for `{‘IM=Auction5443FF’, ‘Role=Buyer’}` to find compatible service implementations for the role she wants to play, which is possible because a service description has at least one pointer to a role it implements for a given IM.
- *Service implementations* Service implementations are pieces of software statically bound to a service descrip-

<sup>1</sup><http://www.openk.org>

<sup>2</sup>for additional information about OpenKnowledge the reader is referred to: <http://www.cisa.informatics.ed.ac.uk/OK/deliverables.html>

<sup>3</sup>[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)

tion. They are typically of average size and consist of a compressed file.

- *Coordinators* Finally, we need functionality to find coordinators. As can be seen in table 1, a coordinator should specify the languages it is able to interpret. Intelligent as opposed to random selection of coordinators for an interaction can have many advantages (e.g. if we select the same coordinator for multiple invocations of the same interaction, we can add interaction participants at runtime).

To the best of our knowledge, there exist no scalable, efficient and fully-distributed implementations for multi-attribute indexing and search. The JXTA project claims to have implemented such a system, but unfortunately, after conducting extensive tests, we discovered that its implementation could not scale to more than a handful of rendezvous peers[7]. The focus of this paper is on scalable, open, efficient and robust publishing and discovery of these resources through a community-supported Peer-to-Peer system.

DHT implementations [10, 12, 1, 9, 2] are currently seen as an important building block for Peer-to-Peer systems for storing content in a completely decentralized way [5]. Nodes function autonomously and collectively form a complete and efficient system without any central coordination. The general idea of DHTs is that each item shared on the network is hashed to a unique key, and that this key together with the content (or a pointer to it) are efficiently routed to the a peer responsible for that key. In this way, each peer is responsible for storing the content (or a pointer to it) that is associated with the key. In principle, all DHT-based systems provide the following functionality: *store(key, object)* storing an object identified by its key, and *search(key)* which returns the object (when it exists) from the peer responsible for the key. Current systems need approximately  $O(\log(N))$  messages to search or store and each peer needs to keep from  $O(1)$  to  $O(\log(N))$  pointers to other peers, where  $N$  is the number of peers in the network [9, 10, 12, 1]. Although they seem to deal very well with key lookups, automatic load balancing and robustness, their application domain is constricted by the absence of efficient methods to search for richly described content.

The discovery service of OpenKnowledge is implemented as a layer on top of DHTs to provide efficient discovery, multi-attribute search and distributed storage through use of the semantics of data being stored. In this paper, we are providing the fundamentals for such a system by presenting a novel popularity-based algorithm for multi-attribute search over richly-described content.

In sections 2 and 3, we position our research followed by its description in 3 We give an evaluation of the system in 4 and we conclude our work in section 6.

1. Auction5443FF {Type=IM, Descr.Term=Auction, Descr.Term=Buy, Descr.Term=Sell, Descr.Term=Dutch Auction, Role=Buyer, Role=Seller}
2. Buyer4325 {Type=Service description, IM=Auction5443FF, Role=Buyer, Descr.Term=Buy}
3. CoordinatorFF32 {Parser=LCC, Parser=BPEL, Certification=Verisign}

**Table 1.** Examples of resource descriptors.

## 2 Multi-attribute search by joining single-attribute searches

It is relatively easy to design a discovery system over a DHT that maintains a distributed inverted index over all attribute-value pairs. Namely, to **insert** a new resource into the system, one could hash each attribute-value pair and store it together with a pointer to the resource in the DHT overlay. Furthermore, to **retrieve** a resource, for each attribute-value pair of a query, a lookup on the hash of this pair is performed and a local join is performed over the results.

We have just adumbrated the design of a simple DHT-based discovery system with perfect recall. Be that as it may, to the best of our knowledge there exists no efficient implementation of such a system. Where does it all go wrong?

- *Distributed join is costly* To perform a distributed join, the initiating peer has to gather all index entries for all the attribute-value pairs in the query. The cost of this can be prohibitively high, especially for queries with many pairs. The problem is further aggravated by their distribution of frequency. In document retrieval systems, terms usually follow a zipf distribution[4] (also see figure 1). Therefore, a query with at least one of these common attribute-value pairs will be too expensive to calculate, since it would imply retrieving all the indexes containing that attribute-value.

Note that it would not be effective to limit the number of entries sent for common terms, since there is no way to know in advance which of these entries are popular. To illustrate our case, consider a query for "Type=Service description"; "IM=Auction5443FF"; "Type=Service description" would appear in hundreds of thousands of descriptions while "IM=Auction5443FF" would appear in only a few.

We can circumvent this problem by storing the entire description to the peers that correspond to the hash of each attribute-value pair. Of course, this comes at the cost of additional storage and bandwidth costs for inserting descriptions. On the other hand, it makes query answering a local operation and querying now costs

only 1 message in the DHT. In the next two paragraphs, we see why this is still inadequate.

- *Load balancing* As previously mentioned, the term frequency distribution often follows a Zipf pattern. This gives rise to severe load balancing problems, considering that a peer responsible for a very common term would have to store a large number of descriptions and process a large fraction of the total queries. To partly alleviate this problem, we can bound the number of descriptions that a peer can store and send queries to the peers where each of the attribute-value pairs. Even so, this would solve the problem only for the querier, since the peers which would have to store popular content would be unresponsive for all keys that are mapped to them. For instance, imagine that the very popular attribute-value "Type=Service description" and rare attribute-value "Descr. Term=Dutch Auction" are both mapped to the same peer. The first will cause the peer to be overloaded and unresponsive. Despite the fact that this may not be a serious problem for the popular term, since descriptions with popular terms are common by definition, it will be problematic for the rare term.
- *Long descriptions* In the previous two paragraphs, we have assumed that descriptions are replicated to all the peers responsible for each of their attribute-value pairs. What if these descriptions are large? It is not unrealistic to assume that they contain hundreds of terms. In this approach, the number of messages and replicas increases linearly with the number of attribute-value pairs in the description, which makes the approach non-scalable.

### 3 Our approach

As a solution to the problem of managing large descriptions and multi-attribute search, this work is focused on popularity-based approaches. The key idea is that popular content is easily available on the network due to high degree of replication. Therefore, we do not need to spend much effort on indexing it, in contrast to rare items. It is intuitively true, and experimentally verified[15], that for very popular items, we need no sophisticated routing, even a flooding approach would suffice. In addition, it has also been verified (looking at the results of previous research) and in [15], that for rare items, additional effort is required.

In [15], the authors suggest that for queries for common items, flooding queries is sufficient, while for rare items, DHTs perform best. Research in the context of the PIER project<sup>4</sup> and in [8] also suggests a hybrid flooding/DHT mechanism (albeit with no efficient way to determine which items are rare). Indeed, for commons terms we are not interested in getting *all* results, if there are millions of them;

<sup>4</sup><http://pier.cs.berkeley.edu>

---

#### Algorithm 1 Rarity-based walk

---

**Require:** A description  $d$  with attributes/values  $(t_1 \dots t_n)$  and identifier  $id$ . Let parameter  $PR$  denote the originating peer set and  $D$  the description set of this peer

**Ensure:**  $d$  is stored.

```

1:  $t := t_m \in (t_1 \dots t_n) | \forall t, |Dt| > |Dt_m| \text{ and } P_t \notin Pr$ 
2: if  $(t = \emptyset)$  then
3:   return
4: else
5:    $Pr := Pr \cup this$ 
6:    $send(d, P_{t_m}, Pr)$ 
7: end if

```

---

**Require:** A query  $q$  for terms  $(t_1 \dots t_n)$ , originating peer set  $Pr$ , the description set of this peer  $D$ .

**Ensure:**  $q$  is forwarded.

```

1: if (enough results found) then
2:   return
3: else
4:    $t := t_m \in (t_1 \dots t_n) | \forall t, |Dt| > |Dt_m| \text{ and } P_t \notin Pr$ 
5:   if  $(t = \emptyset)$  then
6:     return
7:   else
8:      $Pr := Pr \cup this$ 
9:      $send(q, P_{t_m}, Pr)$ 
10:  end if
11: end if

```

---

a hundred would be enough. On the other hand, for rare terms, we are interested in *all* results. Nevertheless, most popularity-based approaches assume prior knowledge of which items are popular which is unrealistic. Our approach is to use statistical information, which is automatically calculated in a distributed way, to determine, on-the-fly, which terms are rare and which queries refer to them, and adapt the routing process accordingly.

An interesting and relevant approach is Mercury[3], supporting efficient multi-attribute and range search using a small to medium-size schema. It relies on hubs, consisting of a collection of peers responsible for storing indexes with a specific attribute, functioning as a sort of sub-overlays. Descriptions are routed to *all* hubs, which means that the number of messages and replicas is proportional to the number of attributes in a description. Furthermore, each peer in the network needs to know at least one peer in each hub, meaning that the number of peer references that need to be kept by each peer is at least proportional to the number of attributes in the system. Needless to say, this approach cannot scale beyond a schema with a dozen of attributes.

In this paper, we focus on index placement and do not investigate caching techniques, or the use of shortcuts to peers that gave good results in the past[14]. The methods proposed in this paper are orthogonal to them and it may be expected that both can benefit from each-other.

The novelty of our approach lies in exploiting the structure in DHTs to extract statistical information useful for routing, with the goal of alleviating the problems with joining single-attribute searches and current popularity-based

approaches. We will describe an algorithm that uses statistical information from the local storages of peers to place descriptors more efficiently. The intuition behind our popularity-based approach is that rare attribute-value pairs are preferred for replication, since: (1) For common attributes-values, it is likely that we will find answers anyway, since more matching descriptors will exist in the system. (2) Rare attributes-values yield a higher information value. (3) Peers responsible for common descriptions are likely to be overwhelmed by descriptions.

To illustrate our case, imagine the following resource description: In the simple approach described in 2, the description ‘Buyer4325’ would be replicated to the peers responsible for ‘Type=Service description’, ‘IM=Auction5443FF’, ‘Role=Buyer’ and ‘Descr.Term=Buy’. It is reasonable to expect that a query for {‘Type=Service description’} would be easily satisfied by many peers. Therefore, it would be a waste of resources and a network hot-spot to replicate the description to the peer responsible for this attribute/value pair (i.e. to the peer where the string ‘Type=Service description’ maps to). On the other hand, ‘IM=Auction5443FF’ is rare, and the description should be replicated to the peer responsible for it, since it would be difficult to find another peer that has this rare attribute-value. But how do we determine whether an attribute-value is rare? Unlike previous approaches, we do not assume external or centralized sources of statistical information, but rely on the properties of the distribution of terms into descriptions and the DHT. So, going back to our example, assume that initially (by random choice) the peer responsible for ‘IM=Auction5443FF’ is selected for replication. It is very likely that this peer will already have descriptions with ‘Type=Service description’ since (a) ‘Type=Service description’ is a very common attribute-value and (b) ‘IM=Auction5443FF’ and ‘Type=Service description’ are semantically correlated. Therefore, in our approach, it should decide not to replicate it to the peer responsible for ‘Type=Service description’ since subsequent queries including ‘IM=Auction5443FF’ would be answered by the peer responsible for ‘IM=Auction5443FF’, while queries including for ‘Type=Service description’ can be easily answered by many other peers (or at least by the peer responsible for ‘Type=Service description’).

Our algorithm is described in natural language in the following paragraphs and formally in algorithm 1.

**Inserting descriptions** Peers pick the attribute/value pair in the description that has the lowest frequency in their own descriptions and has not been used for forwarding. If such an attribute-value pair exists, they mark that it has been used and use its hash-value to map the description on a peer in the DHT.

**Querying** For each attribute-value in the description, the hash-value is calculated and the query is routed to the peer

in the DHT to which that value corresponds. However, if enough answers are found on the peers en-route, the message is not routed further towards the destination peer (according to the DHT routing algorithm) and the query process for that attribute-value pair stops. This is meant to protect peers to which popular attribute-value pairs map to.

Compared to an algorithm that replicates according to attribute-value pairs chosen at random, our approach has negligible additional computational costs, as both determining which are the rarest terms in a description and maintaining a list of term frequency is very fast.

Furthermore, it is interesting to note its anytime behavior. In the beginning, when peers have no overview of which terms are rare, they will replicate descriptions to *all* peers. As the number of descriptions in the system grows, so will the local knowledge in each peer about which attribute-values are popular since peers can approximate the popularity of an attribute-value by counting the occurrences in their own data.

## 4 Evaluation

In this section, we evaluate our approach against an approach that replicates according to attribute-value pairs chosen at random.

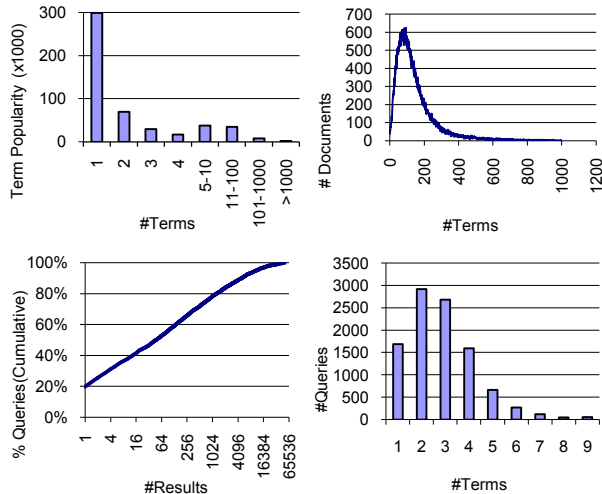
### 4.1 Dataset

Since, to the best of our knowledge, there exists no large dataset for resource descriptions for service workflows/interaction models, we decided to use a dataset created for general-purpose information retrieval, developed for [13]. We believe that this is a realistic assumption because, in a discovery setting attribute-value pairs in a description are semantically correlated. Future research should give additional insight on attribute-value distribution. For now, we assume that it is similar which the distribution of terms in documents. Our dataset was created by crawling a large number of real user queries from SearchSpy<sup>5</sup> and applying a natural language processing method on the results retrieved for these queries using Google<sup>6</sup>, to get relevant descriptions. The input to our system was derived from the following:

- **Corpus** We have used a corpus of 260.000 documents, resulting in the same number of descriptions. Each description consists of a set of terms.
- **Descriptions** From these descriptions, we have selected a random set of 100.000. On average, each document contained approx. 104 terms (the distribution is shown in fig. 1). The distribution of terms, as expected, follows a zipf-like distribution (fig 1). It is interesting that more than half of all terms appear only

<sup>5</sup><http://www.infospace.com/info.xcite/searchspy>

<sup>6</sup><http://www.google.com>



**Figure 1.** **Top left:** Term popularity. For example, around 280.000 terms appear only once in the dataset, and around 139.000 appear 2 times. **Top right:** Distribution of the number of terms per document. **Bottom left:** Number of terms per query. **Bottom right:** Number of results per query (cumulative). We can see that for approx. 50% of the queries, we have less than 50 results and for 30% of the queries, we have less than 4 results.

1 time, while 1 term appears in more than half of the descriptions (58204 times).

- **Queries** To generate queries, we have used the following method: (a) Randomly pick the number of terms  $|t|$  for each query (according to the distribution in 1 (bottom right)). (b) Pick at random a description out of the corpus. (c) Pick  $|t|$  terms (randomly using a uniform distribution) from the chosen description and use them as the query terms.

Figure 1 (bottom right) shows the number of answers per query; for most queries, there are fewer than 50 answers.

## 4.2 Criteria

In this paper, we will evaluate our system in terms of *description recall*. To gain additional insight, we will always take into consideration the number of answers in the system, but limit the number of answers we are interested in to (e.g. maximum 50 answers, which is a parameter that can be changed for each query). The reason behind this, is that, in a discovery setting, there is no point in trying to retrieve *all* answers. Instead, we are interested in getting *enough* answers to satisfy the user. Nevertheless, this is not a limitation of our algorithm itself, it is a choice to make our evaluation more realistic. Thus, for our experiments, recall is defined as follows:

$$D_{Recall} = \frac{|D_{returned} \cap D_{relevant}|}{\min(|D_{relevant}|, 50)}$$

## 4.3 Design, implementation and experimentation

**Design** Our system is based on a three-layer architecture, the bottom layer consisting of a DHT implementation. The second layer consists of an object store and a distributed index supporting multi-attribute search and relies on the algorithms described in 3. Finally, the third layer is application specific, in our case the OpenKnowledge service and peer discovery.

**Implementation** We have implemented our system using Java 1.5. For the bottom layer, we have used the FreePastry DHT implementation, version 2.0b<sup>7</sup>. The second layer is an implementation of the algorithm in 3 and the random approach. The application on top is the discovery service of the OpenKnowledge system[6], as described in the introduction of this paper.

**Testing and experimentation** We have used the DAS-2 distributed supercomputer<sup>8</sup> to test and evaluate our system. One node on the DAS-2 acted as a bootstrap, being used as an access point to the system for the rest. We have used Globus<sup>9</sup> to start 500 instances of our system, which contacted the bootstrap node, and self-organized according to the Pastry protocol[12]. This process took less than 5 minutes. Next, nodes published in parallel 200 descriptions each (100.000 in total). Finally, each node made 100 queries and collected the results.

We have compared our approach to one that replicates descriptions according to a subset of its attributes/values, chosen randomly. For our evaluation, and to have a reference point, we have chosen replicate according to a maximum of 10 attributes/values, thus maintaining 10 index replicas. An approach that would replicate according to all terms would offer perfect recall. Nevertheless, it would require an average of 104 DHT messages and index replicas for each description, which is not scalable. Subset replication does not offer perfect recall, but it does reduce the number of replicas for each description by a factor of 10. Fortunately, as we will see, it does not lead to a proportionate reduction in recall.

For our rarity-based walk algorithm we have adjusted  $Dt_m$  to get the same number of messages and replicas as the subset replication approach. Moreover, for querying, we have used the same policy for both approaches. Therefore, the network, computational and storage costs for the rarity-based walk and the subset replication are very similar.

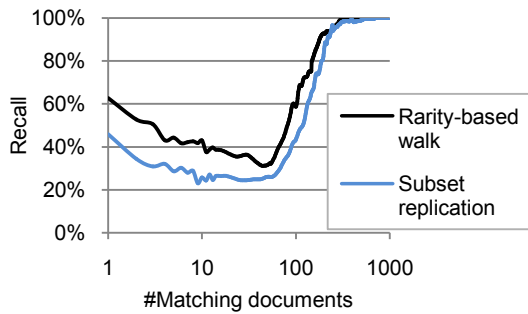
## 5 Results and discussion

Figure 2 shows the results for the two approaches. Each approach required the same number of query messages,

<sup>7</sup><http://www.freepastry.org>

<sup>8</sup><http://www.cs.vu.nl/das2/>

<sup>9</sup><http://www.globus.org/>



**Figure 2.** Description recall as a function of the number of matching descriptions per query.

an average of 2,5 DHT messages. The first impression is that, for both approaches and for queries with more than 300 matching descriptions, we get almost perfect recall using ten times less replicas and messages compared to an approach that would replicate according to all attributes/values. Our rarity-based walk yields a recall in excess of 60% for queries with only a single matching description, which are the most difficult to answer, outperforming the subset replication approach by 50%. In total, even for a relatively small overlay of 500 peers, we gain a substantial increase in recall using the same number of messages. It is expected that as the network size grows, the recall of the subset replication will deteriorate faster than that of our rarity-based walk approach, since the load balancing problems of the former will be augmented. At the moment of writing this paper we are expecting that in the following weeks, the DAS-3 supercomputer<sup>10</sup> will be made available, providing substantially greater computational power for experimentation. Furthermore, we have developed a Grid-based simulator using the Ibis middleware<sup>11</sup>. With these tools we can verify the aforementioned claim.

## 6 Conclusions

In this paper we outline the functionality and design of a peer-to-peer system to discover service descriptions, service implementations, workflows and peers. We provide an implementation that incorporates a novel algorithm that reduces the scalability problems of multi-attribute indexing and search in DHT networks by automatically calculating attribute/value popularity and using it to reduce the degree of replication. Our implementation was tested by emulating 500 instances of our system on the DAS-2 supercomputer. The results indicate that an algorithm that takes attribute/value popularity into consideration for routing outperforms an algorithm that does not. Future work lies in

testing how this gain in performance changes as the network size increases. Besides this, the robustness of our system toward a high peer churn rate needs to be tested.

## References

- [1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Puceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.
- [2] M. S. Artigas, P. G. López, J. P. Ahulló, and A. F. Gómez-Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *Peer-to-Peer Computing*, pages 49–56. IEEE Computer Society, 2005.
- [3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In R. Yavatkar, E. W. Zegura, and J. Rexford, editors, *SIGCOMM*, pages 353–366. ACM, 2004.
- [4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM'99 conference*, pages 126–134, New York, USA, March 1999.
- [5] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *IPTPS '03*, February 2003.
- [6] A. P. de Pinninck, D. Dupplaw, S. Kotoulas, and R. Siebes. The openknowledge kernel. Technical report, Openknowledge consortium, 2006.
- [7] S. Kotoulas. Extracting and incorporating keyword semantics in DHTs, MSc thesis. Vrije Universiteit Amsterdam, 2006.
- [8] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid p2p search infrastructure. In *In Proc. IPTPS*, 2004.
- [9] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world, 2003.
- [10] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.
- [11] D. Robertson. Multi-agent coordination as distributed logic programming. In B. Demoen and V. Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 416–430. Springer, 2004.
- [12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.
- [13] R. Siebes. pnear - combining content clustering and distributed hash tables. In *Proceedings of the IEEE'05 Workshop on Peer-to-Peer Knowledge Management.*, San Diego, CA, USA, July 2005.
- [14] G. Skobeltsyn and K. Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *P2PIR '06: Proceedings of the international workshop on Information retrieval in peer-to-peer networks*, pages 33–40, New York, NY, USA, 2006. ACM Press.
- [15] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of the IEEE INFOCOM conference*, San Francisco, CA, USA, march 2003.

<sup>10</sup><http://www.cs.vu.nl/das3/>

<sup>11</sup><http://www.cs.vu.nl/ibis>