

Extracting and incorporating keyword semantics in a DHT-based discovery system



Spyros Kotoulas
August 2006, Amsterdam

M.Sc. Thesis

email:kot@few.vu.nl
Department of Computer Science,
Vrije Universiteit Amsterdam,
The Netherlands

Advisors: prof. F. van Harmelen, dr. R. Siebes, prof M. van Steen

Contents

1	Introduction	1
1.1	Terminology	3
2	Extracting Term Semantics	5
2.1	Semantics from one peer	5
2.2	Semantics from more than one peer	6
2.3	Interdependencies with the discovery system	7
3	Statistics-aware discovery	8
3.1	Simple DHT-based discovery	8
3.2	Why are DHTs not enough?	9
4	Policy Breakdown	11
4.1	Description Manipulation	11
4.1.1	Push-based	12
4.1.2	Pull-Based	13
4.1.3	Description Agglomerators	15
4.2	Peer Selection	15
5	Multi-Attribute search: The case for a hybrid semantic/DHT based overlay	17
5.1	Principles	18
6	Settings	21
6.1	Setting 1 - Use the DHT as a distributed index	21
6.1.1	Inserting Descriptions	21
6.1.2	Querying	21
6.2	Setting 2 - Replicate whole term-set	21
6.2.1	Inserting Descriptions	22
6.2.2	Querying	22
6.3	Setting 3 - Replicate only on a subset	22
6.3.1	Inserting descriptions	23
6.3.2	Querying	23
6.4	Setting 4 - Forward queries/descriptions	23

6.4.1	Inserting descriptions	23
6.4.2	Querying	24
6.5	Setting 5 - Walk the descriptions	24
6.5.1	Inserting descriptions	25
6.5.2	Querying	25
7	JXTA	27
7.1	Design Goals	27
7.2	Design and Implementation Pillars	28
7.3	JXTA Discovery Service	29
7.3.1	A loosely consistent DHT	29
7.3.2	Advertisements	30
7.3.3	Shared Resource Distributed Index	31
7.3.4	Queries in JXTA	32
8	Evaluation	33
8.1	Dataset	33
8.2	Criteria	36
8.3	Implementation	37
8.3.1	Settings	38
8.3.2	Process	39
8.3.3	Setup	40
8.4	Results	41
8.5	Fallacies and pitfalls	45
8.5.1	Stale documentation and examples	45
8.5.2	Inadequate code documentation	47
8.5.3	Obnoxious bugs	47
8.5.4	Erratic vices	48
8.5.5	Transient Errors/Unreliable	48
8.5.6	RPV Convergence	48
8.6	Conclusions	50
9	Future Directions	51
9.1	Large-Scale Simulations	51
9.2	Explicit semantics	51
9.3	Going beyond subset matching	51
9.4	Specifying what can be approximated	52
10	Summary	53

Abstract

The focus of this thesis is on peer-to-peer systems where peers describe their services and data by sets of terms. The contribution of this paper is twofold: First, we propose a distributed system that efficiently calculates statistics about these terms. Second, we use these statistics to replicate the term sets among a selection of peers in order to enable clustering and improve the peer selection process. As a side effect, we expect that this process will have a positive influence on the performance of the statistical analysis. We perform an evaluation of the JXTA peer-to-peer infrastructure, focusing on scalability and discovery, revealing a series of programming flaws that lead to grave performance issues. Furthermore, we implement, deploy and evaluate a part of the proposed system using the DAS-2 supercomputer. We use five settings to experiment with our discovery platform, and the best of them yields a large increase in the query throughput of our system, while preserving or increasing recall, compared to the naive approach.

Chapter 1

Introduction

In recent years, a wide variety of resource discovery systems have been proposed and developed. Most of them, like UDDI[35] and Napster[22], support attribute-based retrieval. Their major problem is that they are centralized approaches, with all the disadvantages that this entails: single point of failure, scalability issues, privacy issues and the possibility for censorship, just to name a few.

To alleviate these problems, many peer-to-peer indexing systems have been proposed. Initially, such systems built on an unstructured distributed index and relied on flooding for the propagation of queries[18]. This approach led to a great number of network messages and thus, could not scale [1].

In the past 5 years, Distributed Hash Tables (DHT) [24][23][31][32] have emerged as an efficient and scalable distributed naming system. Nevertheless, they do not provide off-the-shelf functionality for approximate answers/queries or queries with multiple terms. In [33], a solution involving using a precomputed LSI [9] matrix was proposed. The disadvantage of this approach is that this matrix should be globally shared, and updating it is prohibitively expensive.

Parallel to these, we have also witnessed the development of semantic overlays [28][27][37]. Yet, such overlays still either do not use sophisticated term semantics [37] or require some precomputed and (sometimes) globally shared knowledge (eg. a distance table derived from an ontology [27] or LSI [28][33]).

Automatically calculating and maintaining such knowledge about semantics proves to be a challenge for the following reasons:

- **Computational resources** Techniques like LSI are computationally expensive. According to [21], circa 1999 the amount of textual information produced yearly amounts to 23-240 Terabytes ($1TB \cong 10^{12}bytes$), and continues to grow rapidly. Obviously, computers are no longer able to locally store all data, let alone process it efficiently. Note that there are systems that appear as a single computer (e.g. centralized search engines like Google), but are made up of thousands of computer, each of them hosting different data.
- **Data locality** In systems that map terms into a space maintained by

a structured overlay using semantic information (eg pSearch), updating the terms that can be mapped is prohibitively expensive, as it requires to take into consideration a representative (thus large) fraction of data and rearrange all terms in space. In unstructured semantic overlays [28], peers will, typically, not have enough information to locally extract enough semantic information. In both cases, there is a need for an automatic and scalable way to extract semantics on large datasets.

On the other hand, in the field of computational linguistics and information retrieval, there has been a plethora of algorithms developed to obtain semantic relationships from large amounts of data [14]. Nevertheless, considering the amount of data that has to be processed, applications are either very computationally expensive or restricted to specific fields.

In our approach, we are assuming that data or services are described through sets of terms and that the system contains a large number of such sets. We are proposing using research methods from the fields of computational linguistics, information retrieval and distributed systems to calculate term semantics and provide a scalable and distributed discovery service.

We are identifying a series of research directions in peer-to-peer discovery systems and focus on DHT-based discovery systems. We are proposing methods to enrich such systems with statistical and semantic information, to improve performance and to facilitate extraction of this information.

We are aspiring to contribute in the following:

- **A distributed method to extract term semantics:** Once we have descriptions clustered in peers, peers can (locally) extract term semantics using statistical information. More precisely, peers will be able to calculate *term generality*¹, computed based on the overlap between term sets containing that term, *compare term generality*, computed based on the containment relation between the respective term sets, and *term similarity*, using one of the techniques found in the literature.
- **An efficient discovery method using statistical and semantic information:** As previously mentioned, current systems cannot provide a self-organizing scalable distributed discovery service, that takes automatically-update term semantics into consideration. We will use the term semantics extracted to optimize peer selection during the discovery process and provide support for approximate answers. Furthermore, in section 5 we will describe a hybrid semantic/DHT overlay that opportunistically routes queries according to DHT or semantic criteria.
- **Implementation of a distributed discovery system:** We have implemented and deployed for experimentation a distributed discovery system based on JXTA discovery. It can discover resources based on relatively short(100-term) descriptions and multiple-attribute queries.

¹In layman's terms, we can define as term generality the more/less specific relationship between two terms

- **Evaluation of five JXTA data distribution and query answering models:** Finally, we evaluate five discovery approaches based on the JXTA platform from the perspective of data distribution. As a by-product, since there is a significant lack of information on the scalability of the JXTA discovery system (previous experiments were limited to 4 peers), we will perform measurements on the off-the-shelf discovery mechanism of JXTA. Secondly, we will optimize this discovery mechanism for multi-attribute search and evaluate our system with five different data distribution settings. We will perform a series of experiments by deploying our system on the DAS-2 distributed supercomputer².

The structure of the rest of the document is as follows: In the following section, we will provide a brief terminology overview. In chapter 2, we will outline a method to extract term semantics in a distributed setting. Following a review of the implementation of a discovery system based on distributed hash tables its pitfalls(chapter 3), we will attempt a decomposition of the main functionality of a discovery system (chapter 4). A hybrid overlay that opportunistically routes messages will be described in chapter 5. We will solidify some of the methods presented by devising 5 concrete settings in chapter 6. In chapter 7, we will describe an existing implementation of a peer-to-peer substrate, the JXTA platform. We will evaluate our approaches as well as the performance of JXTA in chapter 8. Finally, future directions along with conclusions will be provided in the last two chapters.

1.1 Terminology

Throughout this paper, we will be using the following terminology:

- **(Semantic) description:** Information is inserted into the discovery system by means of **descriptions**. These descriptions are made up by a set of terms, along with any optional identifier(s).
- **Term:** A term is a string in any language/alphabet. For instance it can be a word, an acronym or a technical term.
- **Query:** A query is a set of terms to be matched against the terms in a semantic description.
- **Pull Descriptions:** Peers actively seek for semantic descriptions that they consider relevant to theirs
- **Register:** DHTs produce a hash value for the keys of objects that are to be stored, for each hash-value, there exists a register, which contains all objects with that hash-value.
- **Peer:** A register peer is a host in the system. It may contain a number of registers.

²<http://www.cs.vu.nl/das2>

- **Responsible Peer:** A peer is responsible for a term if it contains the register for that term.

Chapter 2

Extracting Term Semantics

A lot of information is conveyed in the descriptions stored on a discovery system, which can be extracted and used for a variety of purposes: calculating result rankings, optimizing the query routing process, term matching and term mapping, just to name a few. To extract this information, we can use tools and techniques developed in the fields of text processing, computational linguistics and information retrieval. In our approach, instead of applying these methods on random subsets of all data, we would like to exploit clustering in the discovery system to improve performance and develop a more scalable solution. Hence follows a short description of how we can use a DHT-based system to extract some simple term semantics, with practically no performance overhead for the discovery system.

Descriptions are stored according to each of their terms. Therefore, the peer responsible for "XML", will contain all descriptions with the term "XML". This provides us with enough information to extract some term semantics using statistical information. Some semantics can be extracted using only local information (descriptions that are stored in the peer), while for others, it may be necessary to contact one additional peer.

2.1 Semantics from one peer

With semantics from one peer, we are assuming that either the peer that is interested in the information is responsible to the relevant term, or that it is willing to route to the peer responsible for the term. In [25] a method for Automatic Word Sense Discrimination is proposed. Words, contexts and senses are represented in Word Space, a real-valued space where closeness corresponds to semantic similarity. From this representation, we can extract the following information:

- **Term generality:** The variance in contexts should give us a metric of how general a term is. We can define our metric for generality to be

proportional to this variance. To be more precise, we define generality as the variance of the sense vectors described in [25].

- **Homonyms:** Different senses of a term are, by definition, homonyms.
- **Term similarity:** In [25] vector coefficients are used as good indications for term similarity.
- **Term popularity:** The peer responsible for a term can calculate how many times this term appears in its descriptions and we can define term popularity as that number divided by the total number of terms in the peer. Note that this measure *does not* precisely calculate how popular is the term in all the data; it is a relevant measure for popularity and its accuracy is conditional on the description distribution policy of the system. Leaping into the subject of the following chapters, consider that we place descriptions in the peers responsible for each of their terms. Now imagine that we need to determine the popularity of any term in the system: all we need to do is to route a query to the peer responsible for that term, and ask it to return the number of instances of the term. In this case, we can accurately calculate the popularity of the term since every description with that term will appear in that peer.

2.2 Semantics from more than one peer

By contacting an additional peer (i.e. to calculate semantics for two terms, we need one responsible peer for each term), we can *compare term generality*: Given a term that the peer is responsible for, we can calculate how more/less general it is compared to any other term in the system. This information can be valuable for the functions of the discovery system: performing query relaxation, calculating result rankings and determining query routing and replication paths can all benefit from it. As a simple example, consider that a query yields no results; we can decide to replace one or more of the terms in the query with a similar and more general term and reissue the query (query relaxation). We emphasize that to achieve that, we need to consider both *term similarity* (see previous paragraph) and *compare term generality*.

As an illustration of how this method would work, assume that we are interested to find out if 'RDF' is more or less general than 'OWL'. Initially, we will route a query to the peer responsible for 'RDF' to return what is the frequency of 'OWL' in its descriptions containing 'RDF' ($f_{OWL|RDF}$) and a query to the peer responsible for 'OWL' to return the frequency of 'RDF' in its descriptions containing 'OWL'. ($f_{RDF|OWL}$). We can define the comparison of generality as follows:

$$generality(x_1, x_2) = \frac{freq_{(x_1/x_2)}}{freq_{(x_2/x_1)}}$$

where generality $(x_1, x_2) > 1$ means that x_1 is more general than x_2 . Note that this generality measure is only appropriate once we have established that 2 terms are similar enough to perform query relaxation. For instance, comparing term generality of 'RDF' and 'Car' would return more or less useless information, since there is little similarity between the 2 terms.

2.3 Interdependencies with the discovery system

We expect that even with the description distribution of a simple DHT-based discovery system (as described in section 3.1), the methods above will provide useful information, both for the discovery system itself and for services dealing with these descriptions. On the other hand, extracting term semantics would greatly benefit from some of the more advanced description manipulation techniques described in chapter 4.1 and especially from the process described in 4.1.3.

As future research, it would be interesting to investigate how we could integrate these semantic information in the hybrid semantic/DHT overlay described in section 5. Furthermore, it could be possible to use this information to create an automatically-maintained hierarchical topology

Chapter 3

Statistics-aware discovery

Traditionally, discovery systems (and more generally, information retrieval systems) have relied on inverted indexes as a straightforward and efficient method for matching requests to resources.

In recent years, a collection of systems, commonly referred to as Distributed Hash Tables (DHT), has been developed[8]. DHTs provide a uniform naming scheme, assigning a random ID to each peer from a large ID space. In the same time, they assign each possible ID to a peer (in most cases, to the peer numerically closest to the ID). They are efficient and effective in storing and retrieving (key, value) pairs. Each key is hashed and mapped into a point in a large ID space and its value is stored at the peer which is responsible for the zone that contains this point. To locate an object, it is sufficient to route to the peer with the zone that contains the hash-value of the object. When a new peer joins the network, it splits an area owned by another peer and takes over part of it (including the corresponding keys). When a peer leaves the network, its area is taken up by its neighbors.

DHTs provide the infrastructure for reaching any peer within a few hops. Although dependent of the particular implementation, simulation results have shown that most DHTs have a message complexity of approximately $\log(N)$ to reach any peer in a network of N peers[23][24]etc. Therefore, they can be used to maintain this "traditional" inverted index, as we will show in the next section.

3.1 Simple DHT-based discovery

In this section, we will highlight the design of a discovery protocol based on an inverted index distributed over a DHT, assuming that we are searching for resources described by sets of terms.

To **insert** a new resource into the system, for each term in its description, we insert an entry using the terms as the key and the resource, or the ID of the resource, as the value.

To **retrieve** a resource, for each term of the query, we make a lookup for

that term in the DHT. We examine the results one by one or, in the case that we have inserted only the ID, we perform a local join.

Algorithm 1 Simple DHT-based discovery

Require: A resource d with terms $(t_1 \cdots t_n)$ and identifier id .

Ensure: Resource d is indexed.

```

1: for  $i := 1$  to  $i := n$  do
2:   DHT.put( $\{t_n, id\}$ )

```

Require: A query q for terms $(t_1 \cdots t_n)$.

Ensure: Every id with $terms_{id} \supseteq terms_q$ is retrieved.

```

1: Set  $terms_{id_i}$  returned $_{1..m}$ 
2: for  $i := 1$  to  $i := n$  do
3:   returned.add(DHT.get( $q$ ))
4: Set  $id_i$  results
5: for  $i := 1$  to  $i := m$  do
6:   if  $returned_i.terms \supseteq q$  then
7:     results.add( $returned_i.id$ )
8: return results

```

In this system, recall is guaranteed to be 100% , assuming no system or network failures.

3.2 Why are DHTs not enough?

We have demonstrated in the previous section how to design 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 terms in the query. The cost of this can be prohibitively high, especially for queries with lots of keywords. The problem is further aggravated by the distribution of the frequency of terms. In document retrieval systems, terms usually follow a zipf distribution[5] (also see figure 8.1). Therefore, a query with at least one of these common terms, will be incredibly expensive to calculate. 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 useful. To illustrate our case, consider a query for "data","Spyros Kotoulas"; "data" would appear in hundreds of thousands of descriptions while "Spyros Kotoulas" would appear in about a dozen descriptions. Nevertheless, this query would still require transferring all entries for "data", which may be several MB.

We can circumvent this problem by storing the entire description to the peers responsible for each term. 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. In fact, querying now costs only 1 message in the DHT.

- *Load balancing* As previously mentioned, the term frequency distribution 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 for each term and send a query message for each of the query terms. This way, queries will be answered as response to the messages for uncommon terms. In the event that a query has only common terms, a large number of results will still be obtainable, and will be, for most purposes, sufficient. Note that there exist load balancing techniques for DHTs[6], but they do have a significant maintenance overhead and, in any case, the benefit from storing these common entries is very small compared to the resources they consume. As an example, in the dataset used in our experiments, a term appears on half of the descriptions, even if we use a large peers as registers for that term, thus counterworking the effects of the steep distribution, we will still have to pay a high cost to maintain all these, practically useless, entries.
- *Long descriptions* In the previous two paragraphs, we have assumed that descriptions are replicated to all the peers responsible for each of their terms. What if these descriptions are large? It is not unrealistic to assume that they contain hundreds of terms. In this case, both storage and network bandwidth become an important obstacle.

In [30], the authors suggest that for queries for common items, flooding queries is sufficient, while for rare items, DHT performs best. Indeed, for commons terms we are not interested in getting *all* results, if there are millions of them; a hundred would be enough. On the other hand, for rare terms, we are interested in *all* results.

In chapter 4, we will try to break down the important policies of a peer-to-peer discovery system. We will use example systems from the literature along with novel ideas. We will substantiate some of these ideas by constructing and evaluating solid settings in section 6.

Chapter 4

Policy Breakdown

We have undertaken the task of decomposing the functionality of a class of distributed systems that share the same basic functionality. Depending on background, these systems may be branded as *peer-to-peer discovery*, *peer-to-peer search*, *peer-to-peer overlays* or *peer-to-peer information retrieval systems*.

A common trait of current systems of this type is that they are either not aware of term semantics (and therefore base retrieval only on the representation of objects) ([35][24][17][19][34]) or if they do ([33][28][10]), updating the (shared) structures that capture such semantics is expensive.

We will perform an analysis of how are descriptions and queries handled from the perspective of existing approaches as well as from possible new approaches. We will solidify some of the novel ideas in this chapter in 6.

4.1 Description Manipulation

Initial p2p approaches [18], did not distribute any kind of index for stored documents. Naturally, this resulted in having to search the entire network. In this section we are making an analysis of how and what kind of indexes we can distribute over the network. Note, that indexes need not refer to documents or descriptions of documents, they may refer to peer or they may be implicit (we do not distribute indexes, we change the organization of peers instead). Since these issues are very strongly connected, we have decided to examine them together.

Peers can acquire descriptions using either a pull or a pushed based approach. Most systems use the former [28][10][19]: Peers advertise their data (or a summary of their data), and other peers decide to remember them or not. Alternatively, one may argue that it would be more efficient for peers to proactively pursue data that will enhance their local knowledge, since they are not bound by then response time restrictions common in processing and forwarding queries.

The terms *push-based* and *pull-based* are similar in meaning to *passive clustering* and *active clustering* in this context. We have chosen the former set,

to emphasize that description manipulation need not lead to clustering, since, there are approaches that push descriptions randomly [19], and hence *cluster* neither descriptions nor peers.

4.1.1 Push-based

Traditionally, information dissemination and peer clustering is achieved through push-based dissemination of information. A plethora of such approaches has been proposed:

In [27], peers advertise their expertise descriptions to their "neighboring", semantically close peers. In [28], peers may also forward advertisements to the neighbors that, they deem, will be most interested in the advertisement. Finally, in [37], an epidemic protocol is used to disseminate information to semantically close peers as well as random set of all peers.

On the other hand, some approaches employ DHTs as an efficient and uniform mechanism to address peers. In pSearch[33] documents are abstracted in a high-dimensional vector, which is mapped into a space maintained by a CAN[23]. In pNear[26], peers are described by sets of keywords. From these keywords, they select a random subset, and forward their description to the peers that are responsible for these keywords. Furthermore, they cluster semantically, using peers recommendations from these peers.

Irrespective of whether we are replicating document or peer descriptions, we are considering the following:

Part of the description

Depending on the nature of the descriptions, they may be large, or contain word of little discriminating value. For example, a natural language description will contain articles, which, most likely should be ignored for indexing and retrieval purposes. Which part of the description should we take into consideration?

All We take the whole description into consideration .

Random We pick a random subset of the terms in the description.

Most important We extract the most important terms in the description, and index according to them.

Rare We extract the most rare terms in the description, and index according to them.

Dissimilar Use the most semantically apart terms in a description. At first thought, it may seem like an unintuitive approach, nevertheless, consider the following example: We have a description consisting of the terms "animal", "rdf", "xml", "semantic web". If we choose to replicate according to "rdf", and "semantic web", we will steer the description toward peers semantically close to the AI field. It will be unlikely for subsequent queries for "animal" to reach peers in the AI field. On the other hand, if we choose

”animal” and ”semantic web”, the description will be stored both in peers close to AI and in peers close to Nature.

Replication point

After deciding which will be the replication criteria (namely, what terms are we going to take into consideration to decide about the placement of a description), we need to determine how we will decide to which peers we want to replicate the description:

Randomly Replicate to a random set of peers. Most primitive peer-to-peer file-sharing systems distributed indexes to a random set of peers. Notably, Gnutella (v0.6 with super-peers), Overnet and Kazaa distribute file indexes this way. Some subsequent approaches also query peers in a random fashion. The epidemic protocol proposed in [37] propagates advertisements to peers from a random subset to decrease peer clustering time. In pNear, to save on DHT messages, a random subset of terms is chosen and the index is propagated to the peers that correspond to these terms.

Dimensionality Reduction (hash) Reduce the dimensionality of the description using a hash-function. In the general case, the description space is extremely large. To be able to use functionality provided by DHTs, we need to reduce and bound this space. A straightforward way to do this, is to use a hash-function. pNear and JXTA use this method to map terms found in a description to the same space used for peer identifiers. Note that this approach takes into account only the textual representation and not the semantics of descriptions.

Dimensionality Reduction (lex. analysis) Reduce the dimensionality of the description using a lexical analysis method. pSearch performs LSI on a representative set of descriptions and maps documents to this, lower-dimensional, real-valued space. To maintain this space, it employs a content-addressable network (CAN); queries and descriptions are mapped into the same cartesian space, and therefore, retrieval is straightforward and efficient. The disadvantage with this approach is that all possible components of descriptions must be known in advance, in order to calculate the mapping. Updating the mapping function is prohibitively expensive as it would require moving all descriptions in the system.

Similar Replicate to the peers that contain similar descriptions. Note that the criteria used to determine similarity widely vary according to approach.

4.1.2 Pull-Based

With the exception of [36], in the vast majority of p2p overlays, peers focus on sending descriptions to (potentially) interested peers. It would be interesting

System	Random	Hash	Lex. anal.	Similar
Gnutella	X			
Epidemic	X			X
pRoute				X
pNear	X	X		
pSearch		X	X	

Table 4.1: The replication criteria in some systems in the literature

to explore the possibility of peers preemptively asking for descriptions close to their own expertise. Research in the area of epidemic networks suggests that combining push and pull dissemination of information results in peers being "infected" faster and with a greater probability. Although we are not interested in sharing knowledge globally, intuition suggests that pushing advertisements will benefit peer clustering.

It is important that peers ask for the most useful descriptions. Nevertheless, it is not obvious what criteria should be used to determine which are these. To this end, we have identified some possible mustering policies:

None Peers do not pull descriptions. Therefore, peers contain descriptions with hash-value close to their own id, and only those descriptions. Note that this does not rule out descriptions being replicated in more than one peer.

Random Peers pull descriptions from other peers chosen randomly. This can be either achieved by keeping a random set of peers, ala [36] or by using a dht (the design of such a mechanism is usually straightforward)

Frequently appearing terms Peers ask for descriptions with terms that appear most often in their own descriptions. Descriptions for which they are responsible themselves are excluded, since they are very likely to have such descriptions already. The motivation behind this choice lies in the **Distributional Hypothesis**[12]: *Words that appear in the same context tend to be similar in meaning*. By picking the *most frequently appearing* terms, we are maximizing the peer's knowledge on the context in which frequent terms appear. In turn, this knowledge can be used to calculate second-order term co-occurrence as a similarity measure. Finally, it improves data locality, since terms that tend to appear in the same query will be stored in the same peer, reducing the total number of peers that need to be contacted.

Most important terms Peers calculate which are the most important terms for them and ask for descriptions with these terms, again excluding terms that they are responsible for. This calculation can be made using one of the many methods in the literature [16][4].

4.1.3 Description Agglomerators

We are proposing a method where peers play the roles of *Description Agglomerators*. They have an evolving set of interests, and they pro-actively pursue expanding their collection on that set. As they collect more descriptions, their interests may evolve, becoming more, or less specialized. The basic characteristic of this approach, it that these interests will be semantically converging.

To determine their interests, peers calculate which are the most important terms for them (they can use a categorization algorithm to this end). Then, they pull descriptions close to their interests using the DHT. For example, a peer responsible for 'Body' can easily detect that 'Heart' and 'Head' are important terms for it. Then, it can ask for descriptions with these terms from the peers responsible for them. Now, it will have a broader view on topics close to 'Heart', and in the same time, 'Body' will become a more interesting term for it. We can apply such a process recursively. Peers compute the most important terms in their local data, ask for more descriptions with that terms and repeat, therefore improving semantic locality. A very desirable aspect of this algorithm is its anytime behavior: at any point, peers are expected to have a consistent set of interests, along with descriptions matching these interests. Furthermore, we can expect that by adjusting the parameters of the algorithm (e.g. the proportion of new to old descriptions), we expect that it will eventually converge. We base this belief on the fact that, asking for more descriptions matching a given interest would only make this interest stronger, once recalculated.

We expect that this technique will improve the performance of language processing techniques that are applied locally (eg LSI) since the dataset where they will be applied will has a more restricted semantic range. Note that it cannot substitute such a technique applied to the whole dataset, it is rather a scalable alternative that decomposes the dataset for the methods that normally are very computationally complex, and thus cannot be applied to large data-sets.

Furthermore, it should improve the efficiency of the discovery system, since now data are semantically instead of lexicographically clustered. In pRoute, we have seen that semantic clustering of peers can increase recall by an order of magnitude compared to a random overlay[28]. We expect that clustering data instead of peers will have a similar effect, since, in both cases, we are clustering sets of related terms.

4.2 Peer Selection

The key to efficient p2p information retrieval is knowing which peer to ask. While DHTs can provide a global naming scheme for documents and peers, the problem usually is that we do not know what exactly we are searching for. Peer selection plays an equally important role to description manipulation in designing an efficient and scalable p2p system. We can identify the following popular policies:

Flooding Queries are forwarded to random peers. A straightforward approach followed by many systems. For instance, in Gnutella (all versions) flooded queries to a large subset of the total peers in the network; unfortunately, this resulted in a very large number of network messages and thus, it was not scalable, inefficient and suffered from low recall. On the other hand, when combined with more complex description manipulation policies, flooding can work efficiently. For example, in [29], semantic clustering of peers with random query forwarding gave recall an order of magnitude higher than a Gnutella-like approach. Similarly, in pNear, once peers were clustered, a random selection of neighbors gave more than 40% recall, with a very small number of messages. Research results so far suggest that flooding of messages is a good idea only if there is another mechanism to refine the set of possible target peers.

Lexicographic We can use the lexicographic representation of terms to efficiently route to a fixed "rendezvous" for that term. This is a commonly used technique in systems based on DHTs, but for the reasons mentioned in 3.2 they cannot be used for a general-purpose, multiple-attribute discovery system.

Similarity Queries are forwarded to the peers with descriptions similar to the query. The similarity criteria vary from approach to approach. In [29] it has been shown, that irrelevant of whether we use a semantic description distribution policy, semantic forwarding of queries results in higher recall with a lower number of messages. Furthermore, we can expect that, by employing this strategy, we can use a less expensive method for description manipulation.

Chapter 5

Multi-Attribute search: The case for a hybrid semantic/DHT based overlay

To exploit the features of the semantic overlay, combined with the DHT, we propose a hybrid overlay, that opportunistically routes queries according to semantic criteria or via the DHT. In contrast with what we have described in the previous section, in this approach, we propose a modified DHT, rather than a system using a DHT as a substrate. It can be considered as a DHT optimized for multi-key search, but without offering the hard guarantees of traditional DHTs.

Focusing on large descriptions, we realize that many of the policies described in chapter 4 have to be enforced *on-the-fly* rather than *a priori*, in the peer issuing the query or publishing the content. This arises from the fact that a random peer does not have enough data to decide whether a term is common, if the description or the query is common and so on.

Such knowledge is crucial to effective retrieval, especially as far as queries or documents that contain terms with varying discrimination value are concerned. To illustrate our case, a query for a very rare (or unique) term, should be routed using the DHT; i.e. a query for {A5213FED433} should be routed using a DHT-like mechanism, and retrieval will be very effective and efficient. On the other hand, imagine a query {I want a tutorial to edit OWL in XML using XMLspy on my computer}¹. Firstly, some words provide practically no information and can be removed using a stop-word list. For example, 'I', 'a', 'to', 'in', 'using', 'on', 'my' are removed, leaving us with {want tutorial edit OWL XML XMLspy computer}. Some of the terms in this query are important, while others, in this

¹OWL is a language written in XML

context 'want', 'edit', 'XML' and 'computer' provide little information compared to the others. How can we detect these words? One possible approach would be to use WordNet[20]; but we stumble on the following: 'OWL', 'XML' and 'XMLspy' do not appear on WordNet. This arises from the fact that WordNet is meant as a general-purpose dictionary.

Besides generality, another problem with most semantic networks like WordNet is that they rely on human users to build and maintain them, which makes this approach costly and requires human expertise.

We propose to use term information, extracted as discussed in section 2. In this case our query would be likely to become {tutorial OWL XMLspy}, since these terms are related to and less general than {XML, edit, computer}.

But unless we perform as many DHT lookups as the number of terms in a query (see 2.2), how can a peer know which terms are common and which not? Intuitively, we expect that the knowledge that the term "my" is common, is widely spread over the network, since all peers should have many descriptions with that term. On the other hand, the knowledge that "XML" is related to "OWL" should lie in a much smaller number of peers.

Therefore, it becomes evident that we need to involve semantics in the DHT routing process itself. Following is a series of principles that such a system should abide to.

5.1 Principles

En-route replication of descriptions Any peer that receives a description, even if it does so in order to forward it according to DHT criteria, should also replicate that description. Notice the similarity with social networks: not only do people forward information to the right direction, they also keep it for themselves and provide answers later on. The underlying motivation, along with the expected impact in parentheses, is threefold:

- Very popular terms will be present and detectable by most peers in the network.(reduce the number of messages, decrease query latency)
- It will not be necessary to actually reach the peer responsible for a very popular term, since its neighboring peers are already very likely to contain information about the popular term.(improvement in load balancing, decrease latency, reduce accuracy of statistics)
- Increase the probability, especially for queries with many answers, that during the DHT forwarding process, enough answers will be found before even reaching a peer responsible for a query term. Note that this comes on the expense of additional storage for peers; considering the low cost of storage media and the small size of descriptions, we do not believe that it is a substantial cost to pay.(increase recall, reduce number of messages, load balancing, decrease latency, increase required storage space)

Forwarding If a peer receives a query for which it has no knowledge about, it has no choice but to forward it according to DHT criteria. Due to the construction of the DHT and the fact that advertisements are also copied locally in their forwarding process, the query will be forwarded to a peer that will be more likely to contain information about the query. This strategy is also best for very rare or unique terms. Depending on how much information they have for the terms of the query, peers may decide to use semantic criteria instead. Again, we can consider the analogy to humans, if we are asked something for which we know nothing about, we are likely to consult an encyclopedia or a dictionary (both of each index material lexicographically). On the other hand, if we have some idea about the subject, we are more likely to ask other people, or consult specialized sources. As an illustration, imagine that we want to know about 'Brobdingnag'², most likely, the best thing we can do is consult a dictionary or an encyclopedia, and surprisingly enough we will find the answer³. On the other hand, if we were interested in 'Distributed Systems', the best approach would be to consult a colleague or a more specialized source.

Description Replication Descriptions should lie *near* all the peers responsible for their terms. Therefore, the description should be "spread" over all its topics in order to increase the chance of being found by searching in any of the semantic neighborhoods (and thus increase recall). Furthermore, rare terms should be preferred to common terms, since they have greater discrimination value, and they are an indication that the description itself is rare.

Term selection for forwarding queries Queries and descriptions should be forwarded toward the peer responsible for the least general terms, as it is likely that the peer responsible for the least general term will also have information for the more general terms. The justification for this claim is straightforward: since descriptions are clustered into peers, for all descriptions in a semantic area, we will have more information about popular terms than about rare terms, since popular terms, by definition, appear more often. In any case, the peers for the least general terms are very likely to have shortcuts to the peers with the most general (and related) terms.

Furthermore, compared with picking a random term, the system will offer more natural load balancing, since the peer responsible for a very general term will not have to process as many queries.

Forwarding descriptions Contrary to queries, inserting new descriptions in the system is not bound by a strict latency constraints. Therefore, it would be more efficient to "walk" descriptions so as to have more information when making decisions on which peers or according to which terms to replicate. For example, consider the description {"XML", "RDF", "OWL", "OWL-S"}, and that

²the region in Swift's Gulliver's Travels where everything was of enormous size.

³in Dictionary.com Unabridged (v 1.0.1) Based on the Random House Unabridged Dictionary, Random House, Inc. 2006

initially, and erroneously, "RDF" is picked as the rarest term. It will be routed to the peer responsible for "RDF". That peer will be very likely to know, that in reality, "OWL-S" is the most specific and rarest term in the set, and will be able to determine that the query should be replicated to the peer responsible for that term as well.

Furthermore, walking the descriptions should facilitate spreading of descriptions over all semantic neighborhoods. Consider description {"XML", "OWL", "Car", "Sportscar", "Ferrari", "Porsche"}; a peer having little to do with any of the subjects may calculate the following popularity vector for this query: {10, 0, 9, 0, 0, 0}, and decide that the description should be forwarded to the peer responsible for "OWL", "Sportscar", "Ferrari" or "Porsche". Assume that it randomly picks "Sportscar" and forwards it to that peer. That peer, in turn, is very likely to know both about "Ferrari" and "Porsche", and make good decisions on whether the description should be forwarded to the peers responsible for any of those terms (and probably decide that it should not, since they are similar enough to "Sportscar"). In addition, the peer responsible for "Sportscar" is also sure that it is not relevant to "OWL", and therefore the description should be also forwarded to the peer responsible for "OWL". In the end, the description is replicated on "OWL" and "Sportscar". If the originator peer would have to decide, it would be quite likely that it would have chosen only terms about cars, and the description would never have been registered in the semantic neighborhood of "XML" and "RDF".

Chapter 6

Settings

Considering the number of design choices presented in the previous subsections, it is impossible to perform an exhaustive search over all possible designs and strategies. Consequently, we made educated guesses on settings that may work well based on previous research. We will leave testing of all possible techniques presented as future research. Therefore, following our analysis of existing and possible new approaches we have picked out a number of interesting settings to experiment on:

6.1 Setting 1 - Use the DHT as a distributed index

A simple p2p indexing system based on a DHT. Similar to the current indexing system of JXTA. The DHT is used as a distributed index and query answering is based on distributed joins.

6.1.1 Inserting Descriptions

For every term $t_1...t_n$ the description, we store the tuple $[t, Description_{ID}]$ in the responsible peer. On a system with unlimited resources and no failures, this would lead to perfect recall.

6.1.2 Querying

For a query $q(t_1...t_n)$, we retrieve all tuples with $t_1...t_n$. We perform a join to retrieve the relevant documents.

6.2 Setting 2 - Replicate whole term-set

Similar to the previous setting, but instead we insert the whole description instead of only the $[t, Description_{ID}]$ tuple.

Algorithm 2 Setting 1

Require: A description d with terms $(t_1 \cdots t_n)$ and identifier id .

Ensure: q is stored.

- 1: **for** $i := 1$ to $i := n$ **do**
 - 2: send($\{t_n, id\}, P_{t_i}$)
-

Require: A query q for terms $(t_1 \cdots t_n)$.

Ensure: q is stored.

- 1: **for** $i := 1$ to $i := n$ **do**
 - 2: send(q, P_{t_1})
 - 3: *performlocaljoin*
-

6.2.1 Inserting Descriptions

For every term $t_1 \dots t_n$ the description, we store the tuple $[t_1 \dots t_n, Description_ID]$ to the responsible peer. On a system with unlimited resources and no failures, this would lead to perfect recall. We need the same number of messages as in the first setting, but the size of these messages is much greater (equal to the size of all terms in the description + the size of the ID).

6.2.2 Querying

It is enough to choose only one of the terms from the query, and send the whole query to the peer responsible for that term. That peer can locally match the query with its stored tuples, and return results with 100% recall.

Algorithm 3 Setting 2

Require: A description d with terms $(t_1 \cdots t_n)$ and identifier id .

Ensure: q is stored.

- 1: **for** $i := 1$ to $i := n$ **do**
 - 2: send($\{t_1 \cdots t_n, id\}, P_{t_i}$)
-

Require: A query q for terms $(t_1 \cdots t_n)$.

Ensure: q is stored.

- 1: send(q, P_{t_1})
-

6.3 Setting 3 - Replicate only on a subset

Although it can guarantee 100% recall with only 1 query message, setting 2 is inappropriate for descriptions with many terms (hundreds). The network

bandwidth and the space required to send and store the hundreds of replicas would be prohibitive. We can use the statistical properties of descriptions to reduce the number of replicas required, on the expense of additional query messages.

6.3.1 Inserting descriptions

A random subset of terms s is chosen, and the description is replicated on the peers responsible for these terms, in a similar fashion with Setting 2.

6.3.2 Querying

Queries are forwarded to the peers responsible for each of the query terms. Results are returned as in setting 2. Note that this approach does not guarantee 100% recall. Nevertheless, it offers a trade-off between expensive insertion (and storage) and expensive retrieval.

Algorithm 4 Setting 3

Require: A description d with terms $(t_1 \cdots t_n)$ and identifier id . Let parameter k be the number of peers to replicate to.

Ensure: d is stored.

```

1:  $s_{1..k} := \text{picksubset}(t_{1..n}, k)$ 
2: for  $i := 1$  to  $i := k$  do
3:   send( $\{s_1 \cdots s_k, id\}, P_{t_i}$ )

```

Require: A query q for terms $(t_1 \cdots t_n)$.

Ensure: q is forwarded.

```

1: for  $i := 1$  to  $i := n$  do
2:   send( $q, P_{t_i}$ )

```

6.4 Setting 4 - Forward queries/descriptions

We can also consider forwarding queries or description indexes to save messages and storage space. We can tap on the fact that rather than being randomly chosen, terms in a description are semantically related, and thus, tend to appear in similar contexts.

6.4.1 Inserting descriptions

Similar to the previous approach, we select a subset s of the description terms. We replicate the description index to the peers responsible for these terms as in settings 3 and 4.

6.4.2 Querying

As in setting 3, queries are forwarded to the peers responsible for each of the query terms. If not enough results are found, queries are also forwarded to the peer responsible for the terms that co-occur the most with the query terms. Note that the best peer to determine this is the peer responsible for the query term.

Algorithm 5 Setting 4

Require: A description d with terms $(t_1 \cdots t_n)$ and identifier id . Let parameter k be the number of peers to replicate to.

Ensure: d is stored.

```

1:  $s_{1 \dots k} := \text{picksubset}(t_{1 \dots n}, k)$ 
2: for  $i := 1$  to  $i := k$  do
3:    $\text{send}(\{s_1 \cdots s_k, id\}, P_{t_i})$ 

```

Require: A query q for terms $(t_1 \cdots t_n)$.

Ensure: q is forwarded.

```

1: for  $i := 1$  to  $i := n$  do
2:    $\text{send}(q, P_{t_i}, \text{this})$ 

```

Require: A query q for terms $(t_1 \cdots t_n)$, 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:    $t := t_m | \forall t, |D_t| < |D_{t_m}| \text{ and } P_t \notin Pr$ 
5:    $Pr := Pr \cup \text{this}$ 
6:    $\text{send}(q, Pr, P_{t_m})$ 

```

6.5 Setting 5 - Walk the descriptions

(We are borrowing 'walk' from the notion of *network walk*)

On systems with a Zipf distribution of descriptions, finding queries that lie in many peers is easy and straightforward; we can just flood queries to the network and the probability of encountering such descriptions is quite high. On the other hand, for rare descriptions, flooding would require a very large number of messages and a large amount of time, if we also consider traffic congestion caused by flooding. On the other hand, DHTs can guarantee efficient routing, but registering all terms of a description in a DHT is very costly. Ideally, we would like a system which uses a cheap mechanism to replicate common descriptions

Setting	#Ins. Mess.	Ins. Mess Size	#Q. Mes.	#Ans. M.
Sett.1	n	1	$O(n)$	$ D(t_1) + \dots + D(t_n) $
Sett.2	n	n	1	$ D(t_1 \dots t_n) $
Sett.3	s	n	$O(n)$	$\Omega D(t_1 \dots t_n) $
Sett.4	hops	n	$O(n)$	$\Omega D(t_1 \dots t_n) $
Sett.5	hops	n	$O(n)$	$\Omega D(t_1 \dots t_n) $

Table 6.1: Cost of each setting in terms of network traffic. We can see the costs associated with the basic operations of the discovery system, abiding to settings 1-5.

and a mechanism providing high recall for rare queries. The previous approaches (especially 1 and 2) can be configured for high recall, but for a cost that is not acceptable. In this setting, we will exploit the fact that a peer responsible for a term, contains much semantic information about this term, namely with which other terms it appears.

6.5.1 Inserting descriptions

Peers forward descriptions to the peer responsible for the terms with the lowest frequency in their descriptions. Although it may be unintuitive at first glance, it has the following characteristics: First, rare terms are favored against common terms, since, by definition, common terms will appear more frequently. Secondly, descriptions will be "spread" across a wide semantic area (also see sections 4.1 and 5). Descriptions are thus walked to peers responsible for diverse and rare terms.

6.5.2 Querying

We can use a similar approach for querying. Initially, the query is routed to the peer responsible for a random term in the query set. If not enough results are found, it is routed to the peer that is responsible for the term that appears the least number of times in that peer's description. In the event that there are still not enough results, similar to setting 4, peers may forward the query to the peers responsible for the terms that co-occur the most with the query term.

Algorithm 6 Setting 5

Require: A description d with terms $(t_1 \cdots t_n)$ and identifier id . Let parameter $hops$ denote the maximum number of hops the description can be forwarded, originating peer set Pr , the description set of this peer D

Ensure: d is stored.

- 1: $t := t_m | \forall t, |D_t| > |D_{t_m}| \text{ and } P_t \notin Pr$
 - 2: **if** $hops > 0$ **then**
 - 3: $hops := hops - 1$
 - 4: $send(q, hops, Pr, P_{t_m})$
-

Require: A query q for terms $(t_1 \cdots t_n)$. Let parameter $hops$ denote the maximum number of hops the query can be forwarded, originating peer set Pr and the description set of this peer D .

Ensure: q is forwarded.

- 1: **for** $i := 1$ to $i := n$ **do**
 - 2: $send(q, P_{t_i}, this)$
-

Require: A query q for terms $(t_1 \cdots t_n)$, 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: $t := t_m \in (t_1 \cdots t_n) | \forall t, |Dt| > |Dt_m| \text{ and } P_t \notin Pr$
 - 5: $Pr := Pr \cup this$
 - 6: $send(q, P_{t_m}, Pr)$
-

Chapter 7

JXTA

The JXTA platform^[17] is currently the most mature generic peer-to-peer infrastructure. It is especially attractive because it abstracts from network protocols, thus circumventing common connectivity problems in P2P networks such as over-restrictive firewalls and NATs.

It boasts a load of very useful features: automatic maintenance of group shared data structures, decoupling of peers from physical hosts, post-boxes from peers that temporarily go off-line, an infrastructure for secure communication and secure groups and more.

On the other hand, there has been limited evaluation of its scalability properties. Previous experiments, with the exception of ^[15] have involved a very small number of peers (i.e. ^[11]). Moreover, as far as the discovery subsystem is concerned, there has been practically no evaluation of its performance (i.e. scalability, robustness). Consequently, we would like to perform a series of experiments on the JXTA discovery system, at the very least to have a baseline for the evaluation of our own discovery system.

In the following sections we will briefly the JXTA platform (version 2.4) and its main design goals. We will focus our description on the discovery mechanisms and the rendezvous network, since they are the most relevant to our experiments.

7.1 Design Goals

JXTA serves as middleware, abstracting from networks protocols and topologies, as well as from peer topology. Furthermore, it provides network instrumentation functionality and a discovery protocol for peers and services. From various published documents concerning JXTA design and implementation ^{1 2}, we have extracted the following goals along with their corresponding design decisions ³ :

¹<http://www-128.ibm.com/developerworks/java/library/j-jxta2/?ca=dnt-445>

²<http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html>

³Please note that not all information presented here appear in these documents; some were subsumed by examining the implementation and documentation of the system

Scalability JXTA aspires to provide scalability in the number of participating peers, services and management of the network.

Performance Use the most efficient communication methods available, to get comparable performance to a system designed directly on top of the transport layer.

Developer Friendliness Design a simple API to hide the complexity of the system from programmers

Transparency Applications running on top of JXTA need not know about which network protocol is being used. Furthermore, they need not be aware of the location of peers, obstacles in the network (such as firewalls and NATs). Finally, the network self-organizes for purposes of infrastructure failures. On the whole, it aspires to offer *Access*, *Location*, *Migration*, *Failure* and *Security* transparency.

Mobility The JXTA platform is mobile in terms of platforms (currently there exist implementations for J2SE, J2ME and C) and in terms of node mobility (peers may switch network protocols and endpoints while they are on-line).

7.2 Design and Implementation Pillars

The JXTA architecture is based on 3 pillars:

Organization in Groups Peers are organized in autonomous peer Groups which are, in turn, organized in a hierarchy. Each group is responsible for maintaining its own services and policies. For instance, each group has its own, independent discovery service and its own security policies along with the mechanisms to enforce these policies (peers running these services are different for each group). Note that a peer may belong to more than one group, possibly having different roles in different groups.

Super-Peer network For each group, network services are maintained by a Super-Peer network, called the *Rendezvous network*. Peers are free to join or leave this network at any time, subject to group policies. There is a core set of services (known as core peer group services) that need to be run in this network, namely:

- Discovery Service, used to find peers, services and groups.
- Resolver Service, Resolves JXTA identifiers into network addresses.
- Membership Service, the mechanisms to implement the policies concerning which peers are allowed to join a group and with what role.
- Access Service, to validate requests from peers (e.g. handling credentials)

- Monitoring Service, monitor the state of peers and communication channels.
- Relay Service, used to propagate JXTA messages using the underlying network protocols, possibly circumventing firewalls and NATs.

The Rendezvous network typically consists of well-connected, powerful peers, although this is not mandated by the protocol; for instance, a super-peer network of a small group may consist only of one, relatively weak, peer. Finally, if there is no rendezvous network for a group yet, any peer that detects that, has the responsibility to become a rendezvous itself.

Implementation-neutrality The JXTA protocols is implementation-neutral, allowing porting to any system, from cell phones to enterprise servers.

7.3 JXTA Discovery Service

We will focus on the reference implementation of the JXTA Discovery Service, since it is most relevant to our research. JXTA comes with some discovery mechanisms implemented. For each group, a single discovery service is maintained which is responsible for discovering any type of resource (Peers, services, groups etc).

7.3.1 A loosely consistent DHT

(Also refer to [34] and [3].)

Traditional DHTs (e.g. [23], [31], [24] etc) rely on the assumption of a network with a low churn rate. Maintaining and managing a host with high availability may be a strong deterrent to contributing resources to a p2p network. The reference implementation of JXTA uses a loosely consistent DHT for the super-peer network, which tries to strike a balance between the high maintenance of traditional DHTs and the inefficiency of unstructured peer-to-peer systems.

There are 3 major differences between the JXTA DHT and traditional DHTs:

- **Global View** In traditional DHTs, peers know a subset of the rest of the peers in the DHT (determined by the structure of the DHT, and of size in the order of $O(\log(N))$ where N is the total number of peers in the DHT). In the loosely consistent DHT, peers keep a global peer view (in JXTA terminology, it is called the Rendezvous Peer View(RPV)). Consequently, one can expect that the loosely consistent DHT is very efficient for small RPVs and networks with small churn rate, since a DHT lookup would cost only 1 message. On the other hand, it can not scale to large numbers of peers like traditional DHTs. JXTA developers recommend somewhere between 100 and 200 peers⁴.

⁴Private correspondence with Mathieu Jan

- **Consistency** Unlike traditional DHT approaches, the Peer View in JXTA needs not be consistent: peers make a best effort to keep a consistent global peer view. Therefore, in networks with high availability, we can expect that peers are able to manage this; in networks with low availability, JXTA falls back to a more expensive crawling approach, described in the following paragraph.
- **Limited Range Walker** Since it cannot rely on the consistency of the RPV, JXTA employs a walker that performs a search in the vicinity of the peer where the ID should be located. A more detailed description is given when we describe DHT lookups.

The mechanism implementing the loosely-consistent DHT is straightforward: Each peer has a unique ID, randomly chosen from a large space(128-bits) using a uniform distribution and keeps a list of all other peers that it knows, ordered according to peer ID. This list is called Rendezvous Peer View and elements in that list are periodically exchanged with other peers. In the following paragraphs, we will describe the main DHT functions.

Peer Joins The new peer is added to the RVP of the entry peer, and copies the RVP of the entry peer. Other peers will be eventually informed about the new peer either by being contacted by the new peer itself, or by the entry peer.

Peer Leaves Peers are assumed to fail or leave the network silently. Failure is detected by periodic pinging and failed entries are purged from the RPV.

Publish Items to be indexed in the DHT are assigned an identifier, determined by a hash function. The identifier space is divided in equal zones according to the number of peers in the RPV, and the item is indexed in the peer corresponding to the appropriate zone. For fault-tolerance purposes, the item is also replicated in the neighboring peers as well.

Lookup The hash function is applied to the query to get the identifier in the DHT. The same process as in publishing is applied to send the message to the peer that should contain the item. If the item is not found, the Limited Range Walker protocol is initiated: The receiving peer propagates the message iteratively upwards and downwards in its RPV until the item is found or a maximum number of hops is reached. As a clarification, the query will be propagated to the peers with nearby IDs(remember that peers in the RPV are sorted according to ID).

7.3.2 Advertisements

Central to the discovery system of the JXTA platform is the use of advertisements, small XML documents containing the specification and/or location of an

Peer 1	Peer 2	Peer 3	Peer 4	Peer 5
1	1	1	1	2
2	2	2	2	3
3	4	3	3	4
4	5	4	4	5
5		5	5	

Table 7.1: The RPVs of 5 peers, with peer 2 having an inconsistent view. For simplicity assume an identifier space of 100 and 1 replica per object. Consider the following example: (a) Peer 1 inserts object with ID=86: In its RPV of size 5, 86 maps to the 5th position, therefore, the item will be replicated to peer 5. (b) Peer 3 looks up object with ID=86: In its RPV of size 5, 86 maps to the 5th position, therefore, it will forward the query to peer 5, which will contain the object. (c) Peer 2 inserts object with ID=61: In its (inconsistent RPV of size 4, 61 maps to the 4th position, therefore, the item will be replicated to peer 5. (d) Peer 1 looks up object with ID=61: In its RPV of size 5, 61 maps to the 4th position, therefore, it will forward the query to peer 4, which will not contain the object. Peer 4 will propagate the query its neighboring peers (3 and 5), the object will be located in peer 5.

object or a resource. They are used to convey information concerning the location of peers, objects, entry points to groups, services, specifications of groups, group descriptions etc. It is beyond the scope of our work to describe all possible advertisement types; therefore, we will limit ourselves to describe the part related to discovery:

Advertisements are mobile descriptions consisting of a unique identifier and a number of fields (e.g. name, endpoint). Some of these fields are indexed (and thus handled by the Shared Resource Distributed Index), so as to optimize search.

7.3.3 Shared Resource Distributed Index

The Shared Resource Distributed Index (SRDI), leverages the DHT to provide efficient search among rendezvous peers. It uses the DHT as an distributed index for advertisement fields. Focused on reducing network traffic, it simply propagates collections of `{fieldID, fieldValue, peerID}` tuples among peers, where `fieldID` is the attribute to which `fieldValue` refers (for example, `fieldID` can be "Name" and `fieldvalue` "OpenKnowledge"). The `peerID` is the ID of the peer where the advertisement resides. We emphasize rather than replication the whole advertisement is *not*, JXTA merely distributes indexes for some of its fields. The propagation mechanism is straightforward: On fixed intervals, peers select all advertisements that they have not propagated through the DHT. For each advertisement, they forward a `{fieldID, fieldValue, peerID}` tuple for each of its indexed fields to the peer

where the concatenation of `fieldID,fieldValue` maps. Otherwise phrased, they publish on the loosely-consistent DHT all tuples with indexed fields, using the concatenation of `fieldID,fieldValue` as a key and the peer as value.

To improve redundancy and performance of the DHT, provided that the rendezvous network exceeds a certain size threshold (currently 3), the whole advertisement is replicated to adjacent peers in the RPV.

We will leave explanation of SRDI lookups in the context of general JXTA querying to be described in the next section.

7.3.4 Queries in JXTA

We will present the methods used by JXTA to answer queries. We are referring only to remote searches, i.e. searches that are commenced after failing to find results in the local cache. Sorted according to cost, JXTA tries the following (in this order):

1. **propagate to LAN via broadcast/multicast:** The query is broadcast or multicast over the local area network, to reap the rewards of data locality and similarity in interests in the local network. The cost is merely a local area broadcast/multicast message, and the expected latency very low.
2. **send to rendezvous:** If no answers are found, the query is propagated to the rendezvous peer of the querying peer. The rendezvous peer may already contain an answer to the query, in which case it returns it and the query process finishes.
3. **check SRDI local entries:** If no answer is found in the advertisement store of the rendezvous peer, the latter consults its SRDI entries. If there exists an entry satisfying the query, it is forwarded to the peer denoted in that entry. This peer is then responsible to contact the querying peer with the result.
4. **map into RPV starting point:** The loosely consistent DHT is used to find the peer which would contain an entry matching the query. Using the mechanism described in 7.3.1, the query is routed to the peer where the SRDI entry should lie. If a matching entry is found, the query is routed to the peer containing the advertisement for that entry.
5. **use limited range walker** As a last resort, the query is walked upwards and downwards from the RPV starting point(also see previous section and table 7.1).

Chapter 8

Evaluation

We will focus our evaluation on the techniques described in section 6. Furthermore, we will use the default discovery mechanism of JXTA as the baseline for our experiments and we will modify and extend it with our algorithms.

8.1 Dataset

We have used the dataset developed for [26]. It was created by crawling a large number of real user queries from SearchSpy¹ and applying a natural language processing method on the results retrieved for these queries using Google², to get relevant descriptions. The input to our system was derived from the following:

- **Corpus** We have used a corpus of 260.000 descriptions. Each document was made up by a list of terms.
- **Descriptions** Out of these descriptions, we have selected a random set of 100.000 descriptions to use in our system. On average, each document contained approx. 104 terms (the distribution is shown in fig. 8.3). The distribution of terms, as expected, follows a zipf-like distribution (fig 8.1 and 8.2), we can see that more than half of all terms appear only 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:
 1. Randomly pick the number of terms $|t|$ for the query using the probability distribution 0.16, 0.29, 0.26, 0.15, 0.07, 0.03, 0.01, 0.006, 0.006 (fig. 8.4).
 2. Pick at random a description out of the Corpus.
 3. Pick $|t|$ terms (randomly using a uniform distribution) out of the chosen description and use them as the query terms.

¹<http://www.infospace.com/info.xcite/searchspy>

²<http://www.google.com>

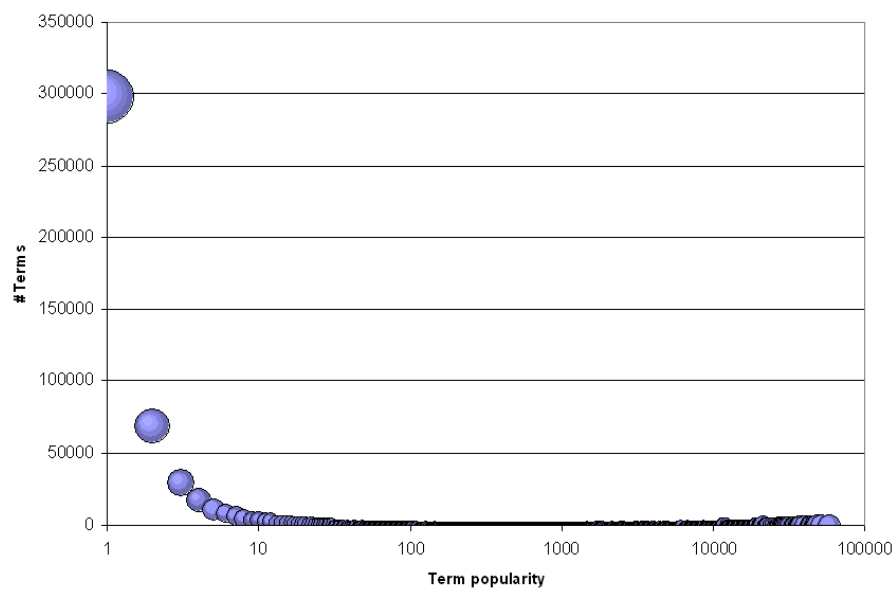


Figure 8.1: Term popularity distribution. We can see the number of terms that appear in the dataset, as a function of their popularity (i.e. the number of documents that it appears in). The size of the bubbles represent the occurrences of the terms in the dataset (essentially, they are the product of the values in the 2 axes).

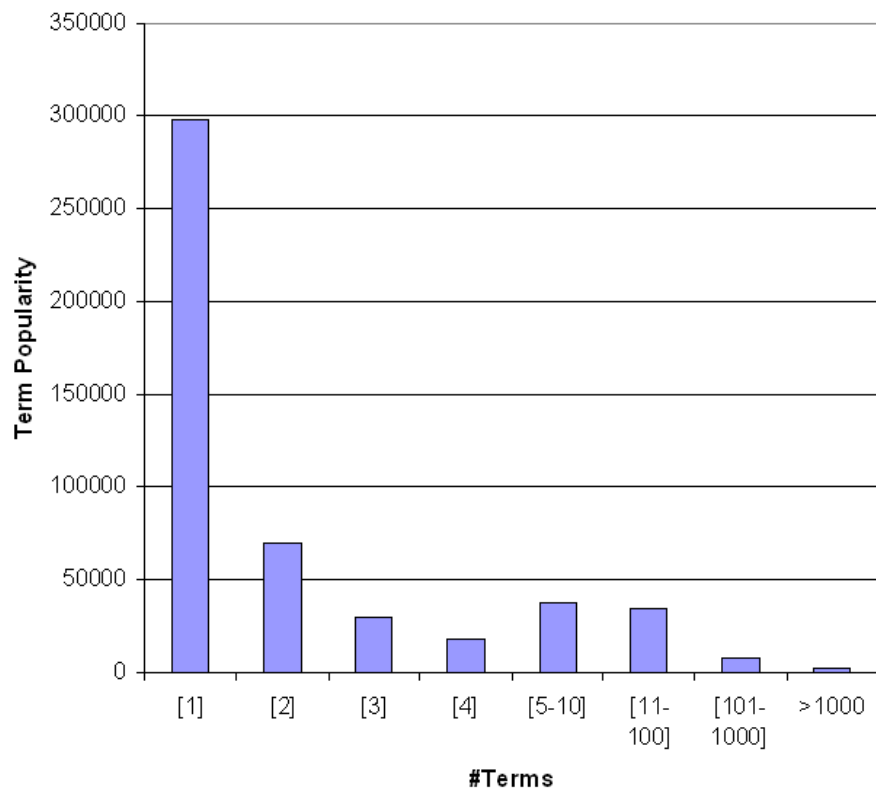


Figure 8.2: Term popularity distribution. Each bar represents how many terms appear the number of times specified in the x axis (for example, the second bar represents that 70.000 terms appear 2 times. Note that most terms appear only once.

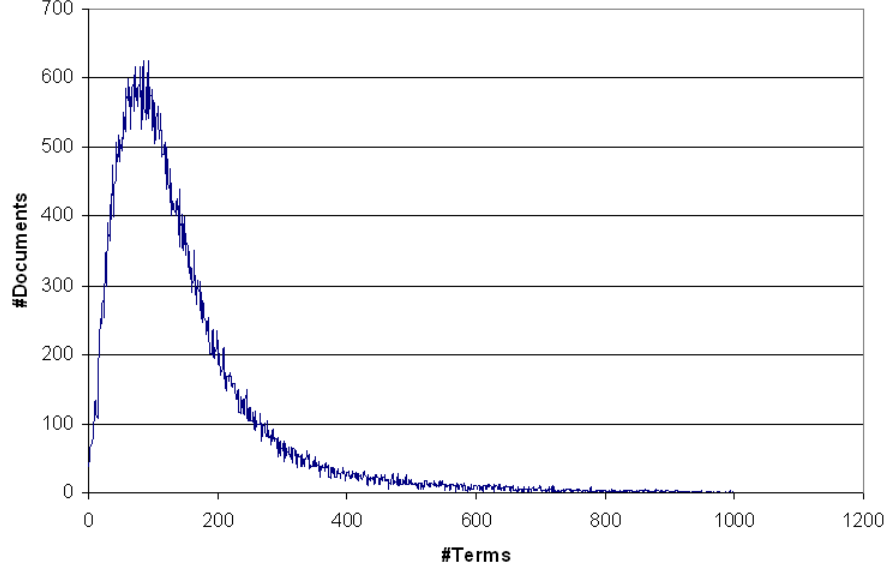


Figure 8.3: Number of terms per document.

Note that not all queries yield results, because some of them are generated by documents from the Corpus that do not exist in the Descriptions.

In figure 8.4, we can see the number of answers per query; for most queries, there are fewer than 50 answers in the dataset.

8.2 Criteria

In order to evaluate our discovery system, we will use description *Recall*, *queries served*, and *query latency*. We consider *relevant* all documents that contain all terms in the query. To gain additional insight, we will always take into consideration the number of answers in the system, and limit them to a fixed threshold (50 answers). Thus, for our experiments, recall is defined as follows:

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

We will also perform a series of load tests, to measure the efficiency of our system. We will measure the total number of *queries served* and *query latency*:

Finally, to evaluate the loosely-consistent DHT of JXTA, we will measure the size of the RPV for each peer.

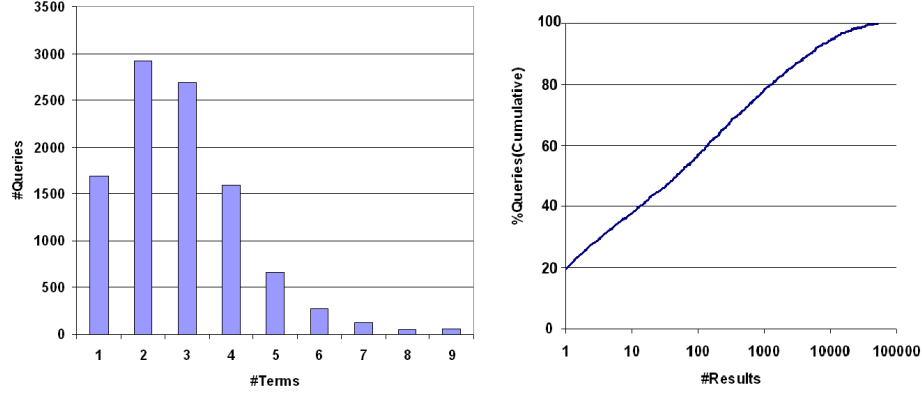


Figure 8.4: **Left:** Number of terms per query. **Right:** Number of results per query (cumulative). We can see that for approx. 50% of the queries, we have less than 50 results and for 30% of the queries, we have less than 4 results.

8.3 Implementation

We provide an implementation of a discovery system implementing the methods described in section 6. We have tested it using the DAS-2 distributed supercomputer. In the following paragraphs, we will outline our implementation, describe our experimental results, issues encountered while implementing the system and guidelines for future research and development of the system.

We have used the JXTA peer-to-peer infrastructure as a basis for our implementation. To isolate our experiments from the global JXTA network, we have created a private (root) group consisting only of rendezvous peers. One peer in that group created a new group, with the customized discovery mechanism. Other peers found and joined that group. Upon joining the group and after waiting for a period of 4 minutes, for the RPV to converge, all peers except for the one that initiated the group, published a number of advertisements containing the descriptions with an interval of 250ms. Following the publication of the advertisements, they waited for 3 minutes, for the SRDI indexes to be propagated. Then all peers posted queries with an interval of 150ms and waited for 5 minutes for results (we did not expect any more messages to arrive after that time frame). Note that we have used the above settings so as to get reasonable results for all experiments. We have experimented with additional settings, but the results were not significantly different, or we were unable to perform some experiments (e.g. setting the intervals too low resulted in peers issuing practically no queries at all).

8.3.1 Settings

In this section, we will highlight the implementation of the settings described in 6 which was based on the JXTA reference implementation. It should be noted that the JXTA architecture and reference implementation can be extended with additional functionality, therefore, our modifications extended rather than modified the JXTA framework.

Setting 1 To implement Setting 1 we have intersected JXTA single-attribute searches. Therefore, for each description, we published a number of advertisements equal to the number of terms in its description. Peers returned all advertisements that match *any* term, and the query issuing peer was responsible to determine which of these advertisements indeed matched the query. In each advertisement ³, we have included the description itself. Note that the description was **not** used for query processing since JXTA does not allow searching according to multiple criteria. The description was only used to tell correct answers (i.e. answers that contained all terms) from all returned answers. This way, we were able to keep description precision to 100%, so as to have comparable results with the other approaches.

Setting 2 We have extended the default discovery mechanism to cater for multi-attribute queries. To achieve this, we had to create a new discovery mechanism to be used only in our group of peers. Note that this implementation is fully compatible with the JXTA architecture and reference implementation, since it allows groups to have customized discovery services. We have enriched the advertisement mechanism with a new field for descriptions with multiple attributes. Queries also modified to support multiple attributes, and they were now matched locally. Thus, peers only returned the advertisements that matched all query terms and matching queries with advertisements was done only at the responder.

For settings 3-5 we have implemented a configuration mechanism to specify strategies to be used by peers. Strategy classes are specified in a settings file, and loaded dynamically during execution. This eliminates the need to ever change the implementation code, for purposes of strategy specification, even if new strategy classes are created. As an example, the directive:

`org.nl.vu.ok.semanticRouting.replicationStrategy.ReplicationStrategy-`

³Initially, we had tried the following approach: to insert a new description, we inserted a new advertisement for each of its terms containing a {term,description-id} pair. To search for a description, we issued a separate query for each query term, and we aggregated the {term,description-id} pairs of these queries. We performed a join of these pairs with the query and got the results. It yielded very bad results for two reasons:

- Failure to retrieve one index for a query meant that the query failed.
- JXTA has a built-in threshold of 50 results.

```
org.nl.vu.ok.semanticRouting.replicationStrategy.Random(4):  
org.nl.vu.ok.semanticRouting.replicationStrategy.Rarest(3)
```

specifies that the description replication strategy is to replicate according to 4 random terms plus the 3 rarest terms. It is possible to create a new strategy class:

```
org.nl.vu.ok.semanticRouting.replicationStrategy.StrangeNewStrategy(3,"penguin")
```

with no change or recompilation of the system, allowing for on-line strategy changes.

Setting 3 For the implementation of the third setting, we have used the aforementioned policy specification mechanism to replicate descriptions according to a maximum of 8 terms, chosen randomly. The search mechanism remained the same as in the previous setting.

Setting 4 Instead of being sent directly to the chosen peers, queries were sent to only one peer that, in turn, forwarded them, assuming that not enough answers had been found. Furthermore, they were forwarded according to the most rare term, but the maximum number of times that they could be forwarded was fixed.

Setting 5 Descriptions were forwarded according to terms with frequency less than 5%. There was no threshold for the maximum number of times a query could be forwarded, but each term can be selected for forwarding only once (terms that have already been selected are marked in the query. Therefore, implicitly, the maximum number of hops for a query was the number of query terms.

8.3.2 Process

To clarify the method followed, we provide a short overview of all steps taken. Note that rather than being a linear process, we had to fall back to the previous steps numerous times, to refine our settings and implementation.

1. Parse the Corpus file.
2. Extract the descriptions to be used and calculate statistics for them.
3. Generate query set and calculate statistics.
4. Define settings and strategies.
5. Run emulation, and write log files for each peer.
6. Process the log files and create statistics for the run.

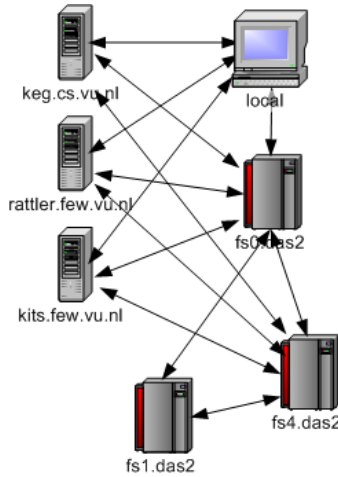


Figure 8.5: Host Configuration. Arrows represent unrestricted network connectivity (no firewall)

8.3.3 Setup

To verify its correctness and to experiment with its behavior, we have deployed our system in a small variety of host configurations (fig 8.5).

local A standard desktop machine (AMD Athlon64 3500+, 1GB memory, Windows XP) running up to 5 instances of peers. We have used this configuration to verify that our discovery system works and produces correct results.

local+servers We have set up a larger experiment involving the local machine plus a number of compute servers available in the faculty, namely:

- rattler and kits (Sun Fire V440z, 4x AMD Opteron 852 2594 MHz CPU, 16.0 GB memory, Debian GNU/Linux)
- keg and fluit (Sun Fire V440, 4x Sun UltraSPARC-IIIi 1281 MHz CPU, 8.0 GB memory), Solaris 9

On the Linux servers, we were able to run up to 50 instances of peers(processes), albeit with a big impact on performance. On the Solaris servers, we could run up to 20 peer instances. This experiment verified, that indeed, our discovery system was able to run on heterogeneous hosts. Nevertheless, as the number of processes per host increased performance plunged to unacceptable levels.

local+das2@vu In this setup, we have used the local computer as an fixed reference point and nodes in the VU cluster of the DAS-2 supercomputer as rendezvous peers. We have tried our system with up to 50 nodes (2xPIII

1Ghz, 1GB memory, Red hat enterprise Linux), running 2 peer instances each. When running more than 2 peer instances per node, they experienced performance problems. Unfortunately, we were negatively surprised by the RPV convergence. As also described in section 8.5.6, the RPV dissemination was not what would be expected from a system running on a fast local network.

local+das2 The DAS-2 consists of 5 geographically distributed clusters. We have run our system using the Amsterdam cluster and 2 more other clusters. Again we have used the local machine as a reference and entry point to the system. What made this experiment interesting was the fact that the nodes in remote clusters (clusters other than the VU-cluster) were protected by a firewall. Consequently, the local machine could not contact them. On the other hand, the local machine could be contacted, since we had requested a port opened on the VU firewall. We have noticed that, contrary to specifications, the system should perform adequately, it took a severe performance blow, and many peers had difficulty connecting at all.

local+das2+servers Our final experiment was to use several DAS-2 clusters along with the compute servers in the VU with the local machine again acting as the entry point to the system. The result was comparable with the previous experiment. Finally, we have decided the using only the VU cluster, with the local machine as an access point, was the only setting that could produce reproducible results with a relatively low performance variation between runs. Even so, in many cases, the loosely-coupled DHT did not perform as expected on several occasions, and suffered from frequent transient failures. Note that all hosts were connected by fast networks; the local machine and the compute servers being connected by Gigabit Ethernet, the DAS-2 nodes in clusters by 100Mbps Ethernet and multi-Gbps inter-cluster connections. To the best of our knowledge, in no occasion was the network congested. Instead, CPU power became the bottleneck.

8.4 Results

In this section, we will present some experimental results concerning the settings previously presented.

In figure 8.6, we can see a performance comparison of the five settings, concerning recall and number of successful queries. In settings 1 and 2, the inefficient description inserting mechanism has a negative effect the total number of queries that can be posted, since peers are busy forwarding thousands of descriptions instead of issuing new queries; therefore, queries are throttled to almost one third of the maximum. On the other hand, recall is not very seriously affected and is on par with the rest of the settings.

An important observation is that for queries with over 1000 answers, all settings performed perfectly, corroborating evidence given in [30] that for popular

items (or equivalently, queries with many results) even a flooding approach is effective enough.

Furthermore, we can see a very big difference in the number of queries that were issued in each setting: In settings 1 and 2, less than half of the queries possible were issued, due to throttling mechanisms cutting back the number of queries posted. On the other hand, in setting 5, almost all queries were posted, and there was an important difference with settings 1-4. The reason behind this is that the term selection algorithm of setting 5 has implicit load adaptation properties: Terms according to which descriptions are replicated are chosen according to their popularity (only terms with popularity below a threshold are chosen). When peers contain few descriptions, many terms are selected because they are not popular enough, increasing the number of advertisements posted in the system. On the other hand, when peers contain many descriptions, and are more likely to be busy, less advertisements will be posted.

Although we can see some difference in performance (especially in the number of queries posted), it is not as significant as we would have expected. We are convinced that this is due to the dysfunctional DHT of the JXTA reference implementation (see also 8.5.6).

Setting 1, as expected, consumed a large number of messages to index descriptions according to all terms. Even more, it performed worse than setting 2 (particularly for queries with few results) because it returned descriptions matching any term. Combined with a maximum of 50 results per query (per peer), even if a peer did contain the correct answer, it may be that that answer was not included in the result set. In 8.7, we can see a scatter graph of recall for queries with different number of answers. As expected, recall increases as the total number of answers in the system grows larger; we notice, that for queries with more than 2000 answers, recall is perfect. We also note, that for queries with 4-100 answers, recall is lower for queries with 1-3 answers. This is due to the query forwarding mechanism of the loosely-consistent DHT: The query is forwarded to the peer where the index should lie, if no answers are found, the query is propagated to the neighboring peers. Even if one answer is found, the query is not propagated; therefore, the system does much more effort to return the first result compared to the effort for subsequent results.

Setting 2 still produced a very large number of advertisement messages. Nevertheless, on the queries that did manage to get posted it has the best recall for queries with few results. Of course, due to load issues, it was still far from the theoretical 100% recall.

In figure 8.8 we can see the distribution of the delay for each answer received. Successful queries required more time, while local hits (hits in few ms) adversely affected recall. This can be explained as that these queries were not preferred for forwarding, since some results were already returned. Elaborating, in figure 8.9 one can see recall as a function of the query latency for each answer as a scatter graph (defined as the time period between the posting of the query and time the answer was received). We can see a tendency to higher recall as the latency of the answer increases. Nevertheless, a number of queries were fully satisfied (recall=1) by the local peer (latency was very low).

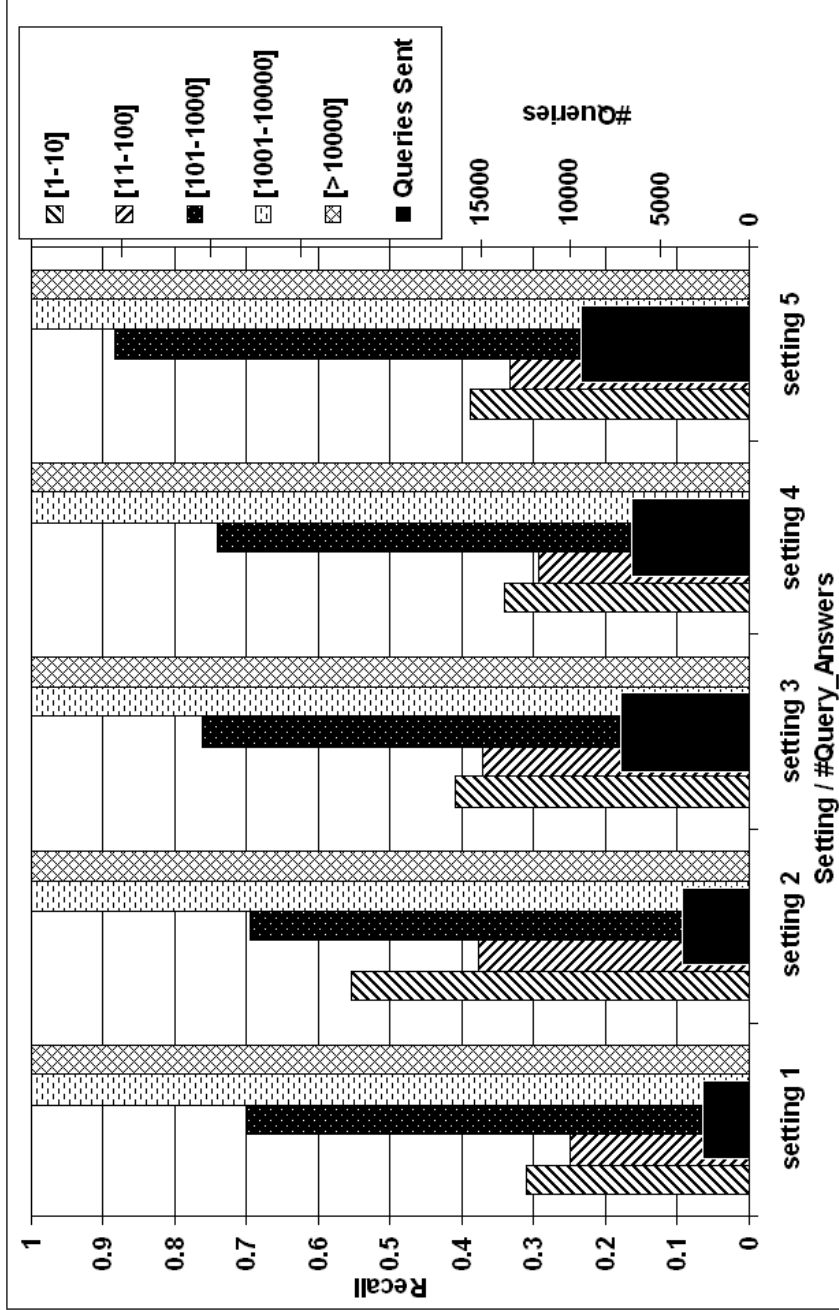


Figure 8.6: Comparison of the 5 settings in terms of Recall. The 5 textured bars for each setting represent average recall for queries with 1-10, 11-100, 101-1000, 1001-10000 and more than 10000 answers. We would like to remind the reader that there is a maximum threshold of 50 answers for each query, thus, if for a query there are 20000 answers and we get back 80, we consider recall to be 1. The wider, black column represents the number of queries that were successfully issued; notice the big difference between settings, being due to the large number of messages used in the first 2 settings to insert new descriptions.

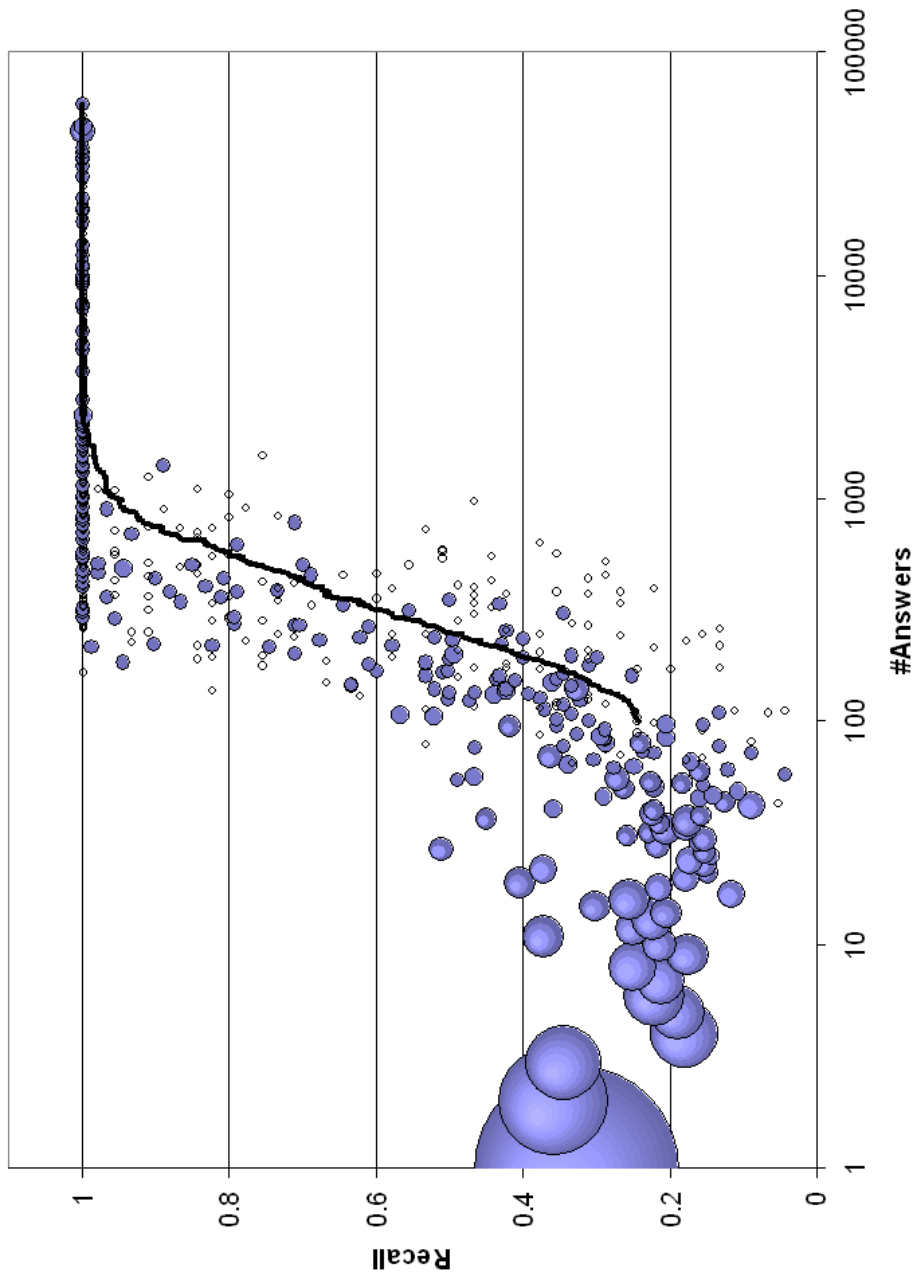


Figure 8.7: Recall as a function of the total number of answers (averaged over all queries with the same number of answers). The size of the bubbles represents the number of samples. For instance, the leftmost bubble signifies that for queries with only 1 result in the system (x-axis), average recall was around 0.35 (y-axis) and there were lots of such queries (size of bubble). The black line is a trend line, depicting the rolling average over a window of 100 samples.

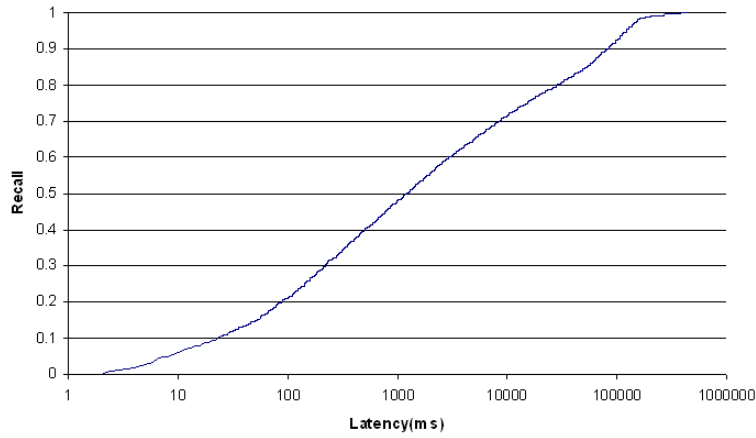


Figure 8.8: Recall as a function of latency, averaged over queries answered in the same time frame.

Finally, in figure 8.10, we can see the average number of returned answers for the queries in setting 3. We can see a peak forming at around 2000 answers per query. We can explain as follows: queries with a very high number of results will be satisfied locally, and will never be forwarded (which could result in gathering more than the threshold of 50 descriptions). Therefore, the average number of returned results is lower. On the other hand, queries with less answers are less likely to be forwarded, and have more than 50 results (queries answered locally will never have more than 50 results).

8.5 Fallacies and pitfalls

Much to our regret, we have come across a number of issues while experimenting with the JXTA platform. In the following paragraphs, we outline the most important problems encountered.

8.5.1 Stale documentation and examples

Although considerable documentation efforts have been made, the most notable being the JXTA programmers guide⁴ and the companion examples⁵, in many occasions, the examples presented contained deprecated code and code that did no longer compile. To make matters worse, the documentation in the JXTA programmers guide was, for a large part, stale, resulting in confusion.

⁴http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

⁵<http://www.jxta.org/ProgGuideExamples.zip>

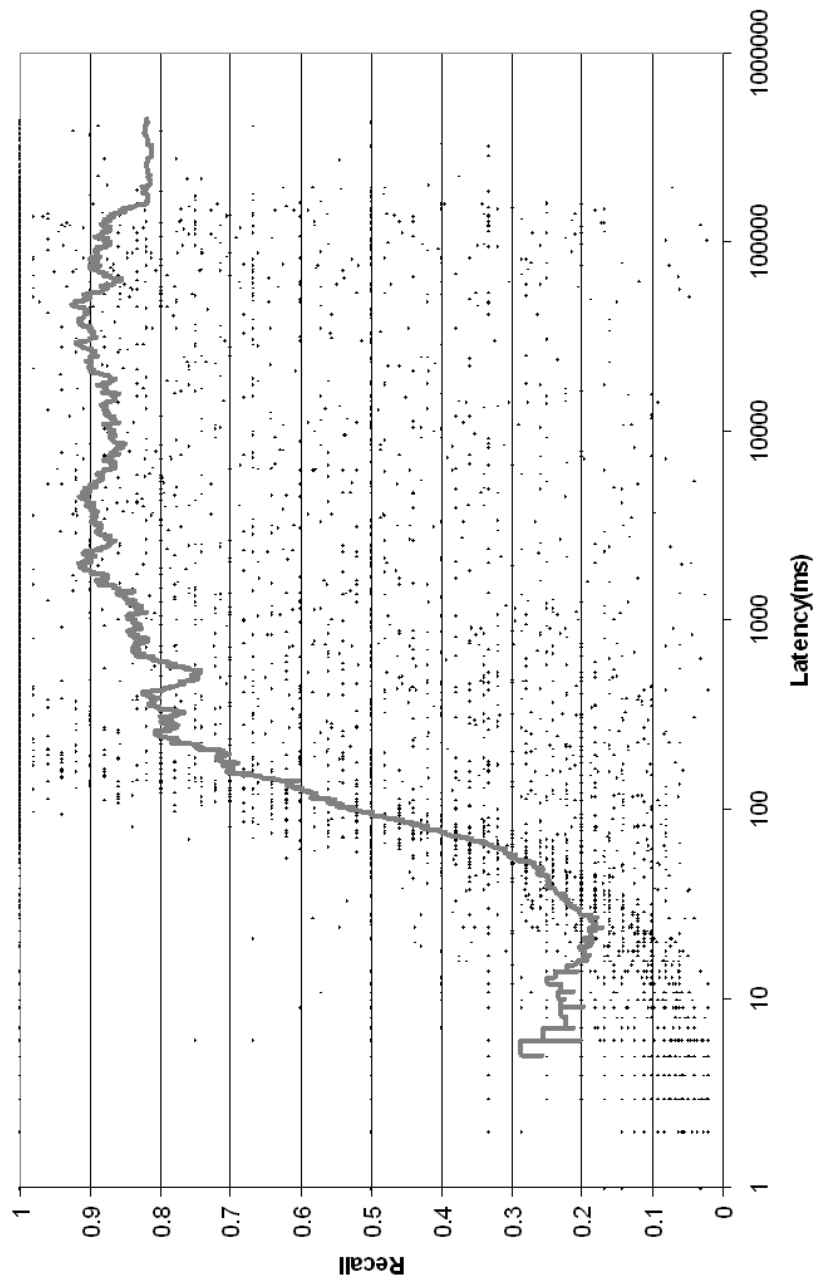


Figure 8.9: Recall as a function of latency. Each dot represents an answer with latency on the x-axis and recall (in total, from all answers) on the y-axis. The line represents the rolling average over 100 samples. Note that more recall values are *not* the average for all queries with the same recall.

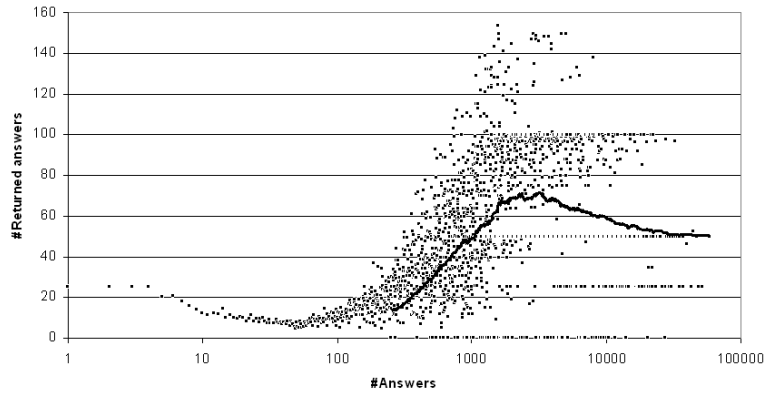


Figure 8.10: Returned answers as a function of all answers in the system. Dots in the graph represent the number of answers in the dataset (x-axis) and the average number of answers returned (y-axis). The line is a trend line of a rolling average with a window of 200 samples. Note that a maximum of 50 answers was requested, and therefore, the ideal value on the y-axis is 50. The cases where we had more than 50 answers are attributed to the query being sent to more than one peer simultaneously.

8.5.2 Inadequate code documentation

Despite the complex architecture of JXTA, on many occasions the documentation of the code was inadequate, misleading or wrong. Especially lacking were the descriptions of how to use classes, information that could only be inferred by the companion example and, mostly, by community effort on various forums. Of course, getting over trivial examples was a laborious task requiring careful examination of labyrinthine code fragments.

In many occasions, we were forced to resort to looking at the implementation to figure out what was the exact function of the methods in the API, although, as described in 8.5.2 and 8.5.3 many times, that did not help either.

8.5.3 Obnoxious bugs

While modifying and extending the JXTA reference implementation, we came across a series of bugs. To our regret, due to the size of the code and the complexity of the architecture, it is difficult to detect, let alone correct, such bugs.

To make matters worse, in some cases the JXTA reference implementation displays a dicey behavior. For instance, we have noticed, that if a peer has more than approx. 600KB (in UTF-8, approx 800 100-word descriptions) of new advertisements, JXTA does not use the SRDI, even when it is manually instructed to push SRDI indexes. What makes the problem very difficult to

tackle, is that, when doing so, it produces no warning or error whatsoever. To circumvent such problems, we have resorted to arbitrary measures, such as forcing an SRDI update after inserting 150KB.

A more amusing, though in reality grave, bug, lies in the following code segment (note that the fact that the only indication that this method does nothing is commented out):

```
/**
 * Specify the Relay enabler.
 *
 * @param isEnabled Relay enabler
 */
public void setRelay(boolean isEnabled) {
// throw new UnsupportedOperationException("this doesn't work");
}
```

8.5.4 Erratic vices

Following the main source of documentation, the JXTA programmer's manual, along with the examples, we have encountered a series of problems that required solutions that were undocumented and seemed arbitrary. For example, the publishing vice:

To create a group, you have to create an advertisement, publish it, discover it and then join the group. Creating a group, and using the advertisement to join the group will not work.

8.5.5 Transient Errors/Unreliable

Finally, the JXTA implementation is tormented by a series of inexplicable transient failures. To animate our argument, in every experiment, some peers (approx 10%) connected to the rendezvous peers, but were unable to discover and join the group. They were issuing discovery requests, but, for some reason, their requests were ignored. This kind of errors make the system unreliable and difficult to benchmark.

8.5.6 RPV Convergence

We have performed a number of experiments to determine the convergence speed of the RPV of the peers in the network. Although all nodes were participating in the same LAN, the RPV did not seem to converge, even after a time frame of 10 minutes.

In figure 8.11, we present the RVP sizes of the peers in one experiment. After joining the network, peers were given 4 minutes for their RVP to converge. Then, descriptions were posted by every peer at a rate of 5 descriptions/second followed by a time period of 4 minutes for the indexes to be distributed. Following, peers posted queries at a rate of 6.66 queries/second. We can see that most peers had

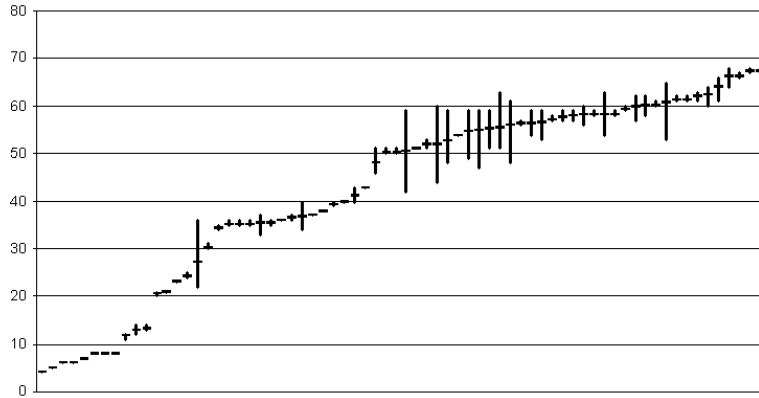


Figure 8.11: Size of the RPV for each peer. The vertical lines represent the minimum and maximum number of peers in the RPV of each peer. The horizontal lines represent the average RPV measured at 4 points: before posting descriptions, after posting descriptions, before posting queries and after posting queries.

a very limited view of the network and no peer seemed to connect to more than 70% of all peers. Note that according to specification, the views of the peers should converge to a shared, global view.

Another problem encountered had to do with network partitioning; we have tried the following scenario:

1. Peer acting as reference point it started.
2. 100 peers are started and find this node, and each other.
3. Peer acting as reference point is forcefully stopped.
4. Peer acting as reference point is started.
5. New peer enters the network, and it finds reference point peer.
6. Neither the new peer or the reference peer are connected again to the 100 peers. This is due to these peers not searching for the reference point peer any more. The network is partitioned and does not recover. Note the IP address of the reference point peer is available through a URL⁶. As it seems, this is a bug rather than a design fault, since in the JXTA protocol, peers periodically check with the referenced peers found in this addressed. They are called the seeding rendezvous, and they are supposed to accelerate RPV convergence.

⁶<http://www.few.vu.nl/~kot/jxtabootstrap.txt>

It goes without saying, that with a dysfunctional RPV mechanism, the loosely-consistent DHT is very unlikely to work effectively. The recovery mechanisms described in section 7.3.1 cannot compensate for so large differences in the RPVs; note that in our experiments, the average RPV size was around 40% of the total peers in the network. Although additional research to determine to what numbers is this implementation of the loosely-consistent DHT capable to scale is needed, our results were discouraging. Unfortunately, to the best of our knowledge, there are no simulation results either.

In turn, this has adversely affected our experimental results, since our methods assume a fully-functional DHT mechanism and we were not able to experiment with large number of peers/physical hosts. On the bright side, we have come to the conclusion, that the tested method do perform well even in the presence of severe failures in the underlying DHT.

8.6 Conclusions

Evaluation our system, we have come to the following conclusions:

- *Increase in number of queries served* Unlike most simulation work, which primarily deals with number of messages and maximum number of hops, we have taken a more pragmatic approach, considering mainly the total number of queries served and query latency. We have seen that with relatively simple modifications to the simple DHT based approach, we can increase the query throughput of our system up to 4 times, without sacrificing recall.
- *Unreliability of JXTA platform* Unfortunately, we came across a large number of problems while experimenting with the JXTA platform, mostly due to the numerous programming errors in the reference implementation and the lack of appropriate documentation.
- *Simulate or use a "bare" peer-to-peer substrate* The existence of a discovery mechanism already implemented in JXTA complicated rather than simplified our own implementation, making it difficult to predict the behavior of the system and to interpret results, since we have not found a detailed specification of the JXTA discovery reference implementation. For future experiments, we would recommend either simulating the system, or using a "bare" DHT implementation (e.g. P-Grid[2], Pastry[24], Freenet[7])

Chapter 9

Future Directions

9.1 Large-Scale Simulations

We have implemented a Grid-Based peer-to-peer simulator on top of the Ibis¹ middleware, able to simulate network latency (and adapt the simulation to actual network latency between simulation nodes) and capable of handling millions of peers. In combination with the DAS-3, which will be available in the following months, we will have the infrastructure to run very-large scale simulations for the methods described in this work (1 Terabyte of main memory and close to 3.5 Teraflops of processing power). Alternately, we can use one of the experimental DHT implementations available (e.g. FreePastry, Bamboo)

9.2 Explicit semantics

It would be interesting to specify directives to explicitly insert semantic information in a peer to peer network. The challenge would be to determine in which peer should this information reside. Apart from the obvious solution of using the hash-table, it might be that in topologies that take semantics into account, it would be beneficial to spread this information to other, potentially interested peers.

9.3 Going beyond subset matching

In this work, we have made the simplifying assumption that we have a query result, when all query terms appear in a description. Current information retrieval research has proposed a plethora of more sophisticated measures. It would be interesting to compare our results against one of these approaches, or even more, incorporate such an approach in our discovery system.

¹<http://www.cs.vu.nl/ibis>

9.4 Specifying what can be approximated

Going beyond sets of descriptive terms, for a full-fledged discovery system we need to specify what can be approximated and how. For instance, if we are searching for computational resources, and we have stored an enormous number of descriptions with "Operating System=Microsoft Windows XP", in no case can we dismiss this information for not having enough discriminating value. It is very unlikely that we will be able to infer whether the information above can be approximated. In most cases, approximating it with "Operating System=Microsoft Windows CE" because they appear on similar context would simply be a bad idea.

One can argue, that the only semantics that we need to be explicit about is whether approximation is appropriate for each term. This should be done both when inserting a description, for purposes of optimizing replication, and when querying. To this end, maybe it would be appropriate to develop a simple query language, specifying whether terms can be approximated, and perhaps, how loose should this approximation be.

It would be interesting to also examine this future work in relation to the PIER project², and especially the language used to query structured peer-to-peer networks[13].

²<http://pier.cs.berkeley.edu>

Chapter 10

Summary

We have proposed a series of methods for multi-attribute search, with a focus on using statistics to optimize replication and retrieval. The methods are not limited to textual descriptions, but can be applied to any content from which we can extract comparable features. Moreover, we have adumbrated their design and the expected challenges and benefits.

We have evaluated the JXTA architecture and reference implementation by deploying it on a relatively large number of computers. To this end, we have performed a series of experiments to evaluate its performance and its adherence to specifications. We came to the conclusion that it suffers from serious programming errors and inadequate documentation.

In addition, we have designed and partially implemented a distributed discovery system that is novel in the following ways:

- It can extract term semantics from a large number of sets of related terms.
- It uses a novel algorithm to replicate data. Peers pro-actively search for descriptions fitting their expertise and adapt their expertise to these new descriptions.
- It employs a new query routing technique that takes into consideration semantic data about terms, when such data is available, and still performs well in the absence of this data.

Compared to systems in the literature, our system is unique in that automatically extracts and updates useful semantic information about data with a low maintenance cost and uses this data to optimize the discovery process.

To evaluate our system, we have performed a series of experiments on the DAS-2 distributed supercomputer. Despite the fact that our deployment was inhibited by failures in the JXTA substrate, compared to a naive DHT-based discovery system, one of our most advanced settings had four times higher query throughput and a moderate increase in recall. The results also indicate tolerance of our methods toward a dysfunctional DHT.

We have concluded with a chapter on additional research that can be carried out in the field.

On the whole, we have provided some new research directions toward multi-attribute search in peer-to-peer systems and have started evaluating them. We are planning to proceed in this research line, elaborating in the methods presented in this paper and deploying them as the discovery infrastructure for the OpenKnowledge project.

Bibliography

- [1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, volume 2172 of *Lecture Notes in Computer Science*, Trento, Italy, 2001. Springer Verlag.
- [2] 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.
- [3] Ahkil and B. T. Project jxta 2.0 super-peer virtual network.
- [4] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Rev.*, 41(2):335–362, June 1999.
- [5] 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.
- [6] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables, 2002.
- [7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.
- [8] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays, 2003.
- [9] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [10] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Olko, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In

- S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2004.
- [11] E. Halepovic, R. Deters, and B. Traversat. Performance evaluation of jxta rendezvous. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, volume 3291, pages 1125 – 1142, 2004.
 - [12] Z. Harris. *The Philosophy of Linguistics*, chapter Distributional Structure, pages 26–47. Oxford University Press, 1985.
 - [13] R. Huebsch, B. Chun, J. Hellerstein, B. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. Yumerefendi. The architecture of pier: an internet-scale query processor, 2005.
 - [14] N. Ide and J. Veronis. Introduction to the special issue on word sense disambiguation: the state of art. *Computational Linguistics*, 24(1):1–40, June 1998.
 - [15] M. Jan and D. A. Noblet. Performance evaluation of jxta communication layers. Research Report RR-5350, INRIA, IRISA, Rennes, France, October 2004.
 - [16] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, London, UK, 1998. Springer-Verlag.
 - [17] Project jxta. www.jxta.org.
 - [18] G. Kan. Gnutella. In A. Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pages 94–122. O'Reilly and Associates, 2001.
 - [19] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network. 3rd IEEE Workshop on Internet Applications (WIAPP'03). Santa Clara, CA., 2003.
 - [20] D. Lin". Review of wordnet an electronic lexical database, 1998.
 - [21] P. Lyman, H. R. Varian, J. Dunn, A. Strygin, and K. Swearingen. How much information?, 2000.
 - [22] Napster. Web Page.
 - [23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM '01*, 2001.

- [24] 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.
- [25] H. Schutze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998.
- [26] R. Siebes. pnear - combining content clustering and distributed hash tables. *Journal on Data Semantics (Springer LNCS)*, 2006.
- [27] R. Siebes, P. Haase, and F. van Harmelen. Expertise-based peer selection in peer-to-peer networks. To appear in: *Journal of Knowledge and Information Systems*, 2006.
- [28] R. Siebes and S. Kotoulas. proute: Expertise-based selection using shared term similarity matrices. In K. Verbeeck, K. Tuyls, A. Nowé, B. Manderick, and B. Kuijpers, editors, *Proceedings of the 17th Belgian-Dutch Conference on Artificial Intelligence*, pages 202–208, Brussels, Belgium, October 2005. Contactforum.
- [29] R. Siebes and S. Kotoulas. proute: Peer selection using shared term similarity matrices. To appear in: *Journal of Web Intelligence and Agent Systems*, 2006.
- [30] 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 Fransisco, CA, USA, march 2003.
- [31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM '01*, 2001.
- [32] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. Technical report, HP Labs, November 2002.
- [33] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.
- [34] B. Traversat, M. Abdelaziz, and E. Pouyoul. A Loosely-Consistent DHT Rendezvous Walker. Technical report, Sun Microsystems, Inc, March 2003.
- [35] Universal Description, Discovery and Integration of Business for the Web.
- [36] S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. van Steen. Exploiting semantic proximity in peer-to-peer content searching. In *10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, Suzhou, China, may 2004.

- [37] S. Voulgaris and M. van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par*, pages 1143 – 1152, 2005.