The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Promotiecommissie:
prof. dr. F. A.H. van Harmelen (promotor)
dr. R.M. Siebes (co-promotor)
prof. dr. Sandro Etalle (Technical University of Eindhoven)
prof. dr. ir. G.J. Houben (Delft University of Technology)
prof. dr. M.C. Rousset (University of Grenoble)
dr. Frank Seinstra (Vrije Universiteit Amsterdam)
prof. dr. M. van Steen (Vrije Universiteit Amsterdam)
Scalable discovery of networked data: Algorithms, Infrastructure, Applications

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op woensdag 24 maart 2010 om 11.45 uur
in het auditorium van de universiteit,
De Boelelaan 1105

door

Spyridon Kotoulas

ergoren te Cholargos, Griekenland
promotor: prof.dr. F.A.H. van Harmelen
copromotor: dr. R.M Siebes
Preface

The idea to start a PhD was neither a long-standing plan nor an epiphany during a late night in a bar. It was suggested to me during the second year of my bachelors. I paid little attention then, but it must have meant something to me, since I still remember the moment. In time, the opportunity came and I started my PhD, which I have definitely not regretted.

They say that doing a PhD is about being able to conduct research independently. I found out that it was much more than that. For me, the most rewarding aspect was communicating and collaborating to exchange and develop ideas. In the end, an idea not shared is an idea eventually lost.

This thesis would have never been realised without the right people. I cannot summarize how much I have learned from the people in the VU, so I will just mention the first associations that come to mind: Ronald (a.k.a. Ronny) guided me in the beginning of my academic life and taught me to work smart. From Frank I have learned to meet smart and be clear. Eyal taught me to act fast and work hard and George to think before I act.

I would also like to thank my new, very promising, partner-in-research, Jacopo, my office-mates Ruud, Rinke and Krystyna, for tolerating my senseless babbling, the people in the Knowledge Representation group and the Web and Media group and the numerous researchers in OpenKnowledge and LarKC. Working with them has been a great pleasure.

On a more personal level, I would like to thank my parents George and Anna as well as my brother Leonidas. Special thanks go to Rika for her creative design. Furthermore, I would like to thank the main passive contributors in this thesis: Kyriakos, Katerina, Minna, Luna and Sotis.

Last but surely not least, I would like to thank my beloved Lithal for her support and I would like to apologize for all the hungry nights she had to endure waiting for me.
Contents

1 Introduction 9
  1.1 Research questions ............................................. 11
  1.2 Summary of contributions ....................................... 12
  1.3 Summary of chapters ............................................ 14
  1.4 Publications .................................................... 15

I Algorithms 17

2 Rarity-based routing 21
  2.1 Introduction ..................................................... 22
  2.2 Multi-attribute search by joining single-attribute searches ... 24
  2.3 Our approach .................................................... 26
  2.4 Evaluation ....................................................... 29
  2.5 Results and discussion .......................................... 31
  2.6 Conclusions ..................................................... 32

3 Discovery of private resources 33
  3.1 Introduction ..................................................... 34
  3.2 Privacy in peer-to-peer discovery systems ....................... 35
  3.3 A framework for discovery of private resources ............... 38
  3.4 A distributed obfuscated index ................................ 39
  3.5 Design choices .................................................. 42
  3.6 Evaluation ........................................................ 45
  3.7 Use case: Openknowledge discovery ............................ 50
  3.8 Application domain .............................................. 51
  3.9 Conclusions ..................................................... 52

4 Scalable rendez-vous 55
  4.1 Introduction ..................................................... 56
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Motivation</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Related work</td>
<td>57</td>
</tr>
<tr>
<td>4.4</td>
<td>Our approach: <strong>SPEEDDATE</strong></td>
<td>59</td>
</tr>
<tr>
<td>4.5</td>
<td>Evaluation setting</td>
<td>64</td>
</tr>
<tr>
<td>4.6</td>
<td>Results</td>
<td>67</td>
</tr>
<tr>
<td>4.7</td>
<td>Conclusion</td>
<td>72</td>
</tr>
</tbody>
</table>

### II Infrastructure

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Going distributed: Peer-to-Peer systems, DHTs and Multi-term search</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Our approach</td>
<td>81</td>
</tr>
<tr>
<td>5.4</td>
<td>Evaluation</td>
<td>85</td>
</tr>
<tr>
<td>5.5</td>
<td>Related work</td>
<td>92</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusions and Future Work</td>
<td>93</td>
</tr>
<tr>
<td>5.7</td>
<td>Acknowledgements</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>96</td>
</tr>
<tr>
<td>6.2</td>
<td>Privacy threats</td>
<td>97</td>
</tr>
<tr>
<td>6.3</td>
<td>Background</td>
<td>98</td>
</tr>
<tr>
<td>6.4</td>
<td>Our Architecture</td>
<td>100</td>
</tr>
<tr>
<td>6.5</td>
<td>Analysis</td>
<td>103</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusions</td>
<td>108</td>
</tr>
<tr>
<td>6.7</td>
<td>Acknowledgements</td>
<td>108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>110</td>
</tr>
<tr>
<td>7.2</td>
<td>Relevant literature</td>
<td>111</td>
</tr>
<tr>
<td>7.3</td>
<td>Motivation</td>
<td>114</td>
</tr>
<tr>
<td>7.4</td>
<td>Our approach</td>
<td>116</td>
</tr>
<tr>
<td>7.5</td>
<td>Performance indicators</td>
<td>119</td>
</tr>
<tr>
<td>7.6</td>
<td>Conclusions and future work</td>
<td>121</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>126</td>
</tr>
<tr>
<td>8.2</td>
<td>Related work</td>
<td>127</td>
</tr>
<tr>
<td>8.3</td>
<td>Our approach: Divide-conquer-swap</td>
<td>129</td>
</tr>
<tr>
<td>8.4</td>
<td>Eventual completeness</td>
<td>130</td>
</tr>
<tr>
<td>8.5</td>
<td>Duplicate detection and removal</td>
<td>132</td>
</tr>
<tr>
<td>8.6</td>
<td>Efficient deductions</td>
<td>132</td>
</tr>
</tbody>
</table>
## CONTENTS

8.7 Experimental results ........................................ 133
8.8 Conclusion ..................................................... 135

### III Applications

9 The OpenKnowledge system .................................... 141
  9.1 Introduction .................................................. 142
  9.2 Relevant literature .......................................... 144
  9.3 An extensive example describing the functionality of the OpenKnowledge system ........................................ 148
  9.4 The architecture of the OpenKnowledge system ........ 156
  9.5 Summary ........................................................ 158

10 Conclusions ..................................................... 161
  10.1 Future work .................................................. 163
  10.2 Outlook ....................................................... 166

Bibliography ....................................................... 171
Chapter 1

Introduction

It is only after we encounter some state of affairs that we find to be desirable that we identify that state as the goal toward which our previous actions, in retrospect, were directed “all along” or “after all”
- Garfinkel, 1967

The Web has been inarguably the most-widely used data networking system. In its original form, it provided links between documents maintained by a set of servers. With the advent of Web 2.0, the ubiquitous Web, the Semantic Web and content producing devices, the Web has changed. It is:

Personal Users have their own space on the Web where they maintain public information (e.g. blogs), semi-public information (e.g. social networking profiles) and private information (e.g. email and documents).

Semantic Some data have explicit semantics (i.e. Semantic Web data). Furthermore, some structured web pages (e.g. Wikipedia pages) have semi-explicit semantics.

Massive Web pages amount to many trillions\(^1\), public Semantic Web triples amount to several billions\(^2\). Semi-public data amount to orders of magnitude more.

From these characteristics, a set of goals for Web Systems is elicited:

Scalability interpreted as the ability of a system to perform as it increases in size. Size may refer to data volume, number of physical hosts or the number of participants. Perform refers to both functional and non-functional properties, e.g. the maximum volume of data that can be handled as well as the access time to it. The need for scalability originates from the massiveness of the Web.

---

1\[http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html\]
2\[http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics\]
CHAPTER 1. INTRODUCTION

Interoperability expressed as the ability of a system to operate under varying data syntax and semantics. The need for interoperability originates from the fact that the information on the web is semantic, thus automated reasoning is possible.

Openness in participation. Needless to say that the Web should be a resource accessible to everybody. Furthermore, the cost for participation, be that financial, in terms of computation or in terms of abiding to certain protocols, should not be prohibitive.

Control of information, participation and resources. Control is closely related to security, privacy and access control. The need from control originates from the fact the the web has become personal.

Completeness in the sense that the system should always produce all results. Note that our notion of completeness is very closely related to the notion of consistency in computer systems [122]. Users interact with the Web: they shop, store personal information, communicate, vote and much more. Since the days when it was used solely as an information medium are long gone, users need to have access to the web and the relevant resources anytime, anywhere.

A plethora of systems and research efforts have attempted to address these goals.

The Semantic Web [10] aims at interoperability and openness by providing a common format and means to specify formal data semantics. Efforts towards open linked data\(^3\) (e.g. DBpedia [12]) transform and link Web 2.0 and legacy data to Semantic Web data, creating a large corpus of data with explicit semantics.

In the area of distributed systems, significant progress has been made in scalability and openness. Modern search engines can scale to tens of thousands of nodes and trillions of web pages\(^4\). Peer-to-peer systems remove the need for central organization allowing open participation and symmetric functionality. Some peer-to-peer systems also offer better consistency than Web systems by keeping data local to the publisher [58] or seamlessly replicating it [68].

The OpenID initiative [94] aims at providing a common authentication infrastructure as means to make control of information more transparent.

Grids aim at scalability and control by providing mechanisms and a unified interface to manage large numbers of nodes spread across multiple organizations [61].

Web Services aim at interoperability through standardization. They provide a uniform way to describe and invoke services on the Web.

More often than not, these goals are conflicting. Supporting more open functionality places limitations on interoperability, completeness becomes more expensive as the size of the system increases, allowing open participation makes control more difficult.

\(^3\)http://linkeddata.org/

\(^4\)http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
1.1. RESEARCH QUESTIONS

To satisfy these goals, some historical compromises have been made, the most prominent being giving up consistency on the Web. Initially, and especially in the field of logics, there were research attempts to keep the web graph consistent. This came into direct conflict with scalability, since it proved to be impossible to keep a consistent graph over a distributed system with no central organization.

Two similar compromises were made in search engines. First, extracting the semantics of each webpage proved to be a very challenging task, so search engines used only textual representations and limited structural information (i.e. hyperlinks). The second compromise regards accuracy: there is neither a guarantee that all webpages are indexed nor that the index is up-to-date.

The focus of this thesis is on discovery of networked data. In this context, discovery refers to data resource location. It differs from search and information retrieval in the sense that it does not deal with ranking results and from network discovery in that it refers to global discovery (i.e. it does not imply a hierarchy in the network or on the resources). In this thesis, several kinds of discovery will be dealt with. Thus, we will defer definition to the next chapters.

Given the explosive growth of online content, we will emphasize on computational scalability. All methods presented are applicable and optimized for large datasets, large numbers of hosts and many participants. Nevertheless, work presented also focuses on at least one of the other goals.

1.1 Research questions

The goals set in the previous section are used to elicit our research questions:

1. How can we design a scalable discovery mechanism for networked data?

2. How does the need for scalability relate to interoperability, openness, control and completeness? Can a good compromise be reached?

We will address the first, design, problem in order to answer the second, knowledge problem.

Our hypothesis is that distribution allows (a) scalable systems and offers a good compromise between (b) interoperability, (c) openness, (d) control and (e) completeness. Among others, this hypothesis is based on the fact that (a) the computational cost can be shared, participating hosts may be (b,c) autonomous, (d) keep control of their own data and (e) maintain their own data. In this thesis, distribution refers to data and computation as well as responsibility.

To validate our hypothesis, we develop a series of methods for scalable discovery of networked data. The methods presented exhibit different behavior as far as the other goals are concerned.
1.2 Summary of contributions

A thematic breakdown of the contributions to the field is as follows:\(^5\):

**Load balancing** Mainly concerned with data or storage load balancing, we have developed two methods that evenly distribute *items* or item descriptors consisting of sets of *terms* among the nodes of a large network:

- **A popularity-based method to balance the entries in an incomplete DHT-based index (chapter 2)**
  
  A key weakness with using DHTs as inverted indexes lies in coping with uneven term distributions. A very popular, an thus not selective term, still needs to be indexed leading to (a) a waste of space and messages and more important (b) excessive load for the node(s) responsible for that term. On the contrary, since it is not selective, its contribution in retrieval will be disproportionately small. We have developed a popularity-based approach that prefers rare terms for indexing and offer a 35% increase in recall compared to a baseline approach. Furthermore, it has no message overhead and minimal computational overhead.

- **A lightweight rendezvous mechanism that balances the number of items per node (chapter 4)**
  
  The popularity-based method described above trades speed for completeness, since it is not guaranteed to find all results. We have also developed an approach that is guaranteed to complete. Being a hybrid between a DHT and a system where peers would randomly exchange items, it relies on data clustering and constant item exchanges. All node maintain exactly the same number of items and process the same number of messages. The intuition behind it is that peer responsibility areas are not fixed and rigid but emerge from the distribution of the data items and oscillate.

**Reasoning** The most challenging part of reasoning with Semantic Web data is locating the premises that lead to some conclusion. In that sense, we can see reasoning as a discovery problem. We will present two radically different approaches to the problem:

- **Backward-chaining reasoning based on a distributed large-scale index (chapter 7)**
  
  Information providers keep their data locally. A DHT-based index mapping terms to information is maintained. Reasoning is goal-driven. Once the relevant terms in a query have been identified, the providers offering data are retrieved from the index and the query is routed to them.

---

\(^5\)hint: The word “scalable” can be added to all titles
1.2. SUMMARY OF CONTRIBUTIONS

The novelty of this approach lies in that there does not need to be a fixed schema or mappings between the schemata of different information providers. Furthermore, data is kept local to the provider, which means that the latter have more control.

- **Forward-chaining distributed reasoning based on the divide-conquer-swap principle (chapter 8)**
  Data is distributed on a set of peers with the same functionality. Each peer calculates the inference of the data it holds and then exchanges parts of its data with another peer. This is repeated ad infinitum. This process is guaranteed to eventually produce the complete inference for all data. We improve performance by applying the rendezvous method previously described so as to increase the chance that data that may produce inference are located in the same peer.

**Privacy** To maintain control in an open discovery system it is essential that information owners can be located without making too much information public. We show an obfuscated index, meaningless to an attacker and a method on top of it that prevents privacy-impairing associations.

- **An obfuscated DHT-based index (chapter 3)**
  We present a method to obfuscate entries in a DHT-based index. Instead of indexing the terms, we index their secure hash. To prevent dictionary attacks, we reduce the size of the hash. Privacy is guaranteed in the sense that this method introduces false positives. The size of the hash is used as a parameter to adjust the level of privacy versus the expense of performance and vice versa.

- **A method to guarantee peer and content privacy (chapter 6)**
  We combine the previously described approach with an anonymizing network to provide additional privacy guarantees. We provide an analysis of the privacy threats in a sharing setting in terms of associations between real identities, the system, content and content descriptions etc. We show a method to offer strong privacy guarantees based on an obfuscated DHT-based index and an anonymizing network. Again this system can support various levels of privacy and anonymity at the expense of performance.

**Resource discovery** Moving up to the most general problem, that of resource discovery, we show three systems based on scalable indexes.

- **Web service discovery over DHTs (chapter 5)**
  We apply the popularity-based routing to Web Service discovery. In collaboration with Seekda inc., we have acquired a corpus of real Web-Service descriptions. These descriptions were decomposed to sets of terms. The
results were very encouraging: We have achieved similar recall with a system based on a full DHT-based inverted index while using only one fourth of the messages.

- A Peer-to-peer interaction system (chapter 9)

The OpenKnowledge system allows knowledge sharing through an open set of peer interactions. The author of this thesis was one of the core architects and developers of the system, with a focus on the overall architecture and the discovery subsystem, where some of the aforementioned techniques were used. The novelty of this system lies in the lack of any centralized component, the lack of a-priori defined semantics and interactions and its scalability.

1.3 Summary of chapters

This thesis consists of three parts:

I: Algorithms

Three algorithms that will be used in the following parts are presented: Firstly, a method to alleviate load-balancing issues and improve performance in DHTs is presented, named rarity-based routing (ch. 2). Its novelty lies in that additional effort is spent on rare items, since popular items are easy to find anyway. Secondly, a method to obfuscate item descriptors in a DHT-based index is described (ch. 3), which allows discovery without having to fully disclose resource descriptions. Thirdly, we show a method to collocate items with the same keys (ch. 4).

II: Infrastructure

In the second part of this thesis, we develop infrastructures based on the methods of the first part. Firstly, we apply rarity-based routing on web service discovery (ch. 5). Secondly, we apply the privacy technique and combine it with a method from the literature to develop a sharing infrastructure with strong and tunable privacy guarantees (ch. 6). Thirdly, we show an architecture to do Semantic Web reasoning in a fully distributed environment where peers keep their data locally (ch. 7). Fourthly, we develop a technique to do distributed large-scale reasoning based on the method to collocate items (ch. 8).

III: Applications

In the final part, the OpenKnowledge system is presented. The OpenKnowledge System (ch. 9) is aiming at scalable knowledge sharing and applies some of the techniques presented in the previous parts for its discovery subsystem.
1.4 Publications

This thesis includes the work in [8, 65, 64, 3, 66, 63, 7, 6] and is also based on work in [113, 100, 99, 112, 38].


• David Robertson, Fausto Giunchiglia, Frank van Harmelen, Maurizio Marchese, Marta Sabou, Marco Schorlemmer, Nigel Shadbolt, Ronny Siebes, Carles Sierra, Chris Walton, Srinandan Das mahapatra, David Dupplaw, Paul Lewis, Mikalai Yatskevich, Spyros Kotoulas, and Adrian Perreau de Pinninck Bas. OpenKnowledge - coordinating knowledge sharing through peer-to-peer interaction. Languages, Methodologies and Development Tools for Multi-Agent Systems, LADS, 2007.

Part I

Algorithms
To efficiently deal with large volumes of networked data, a scalable method to route and place data or pointers to it is required. Typically, routing and placing data is referred to as content-based routing while routing and placing pointers to it as a distributed indexing. In the first part of this thesis, three algorithms for data placement are described:

Chapter 2 refers to creating a scalable distributed index using a rarity-based technique. Chapter 3 describes methods to maintain privacy in such an index. Chapter 4 describes a scalable and load-balanced method to collocate items with the same descriptor.

All chapters in this section deal with data or storage load balancing in datasets with highly skewed popularity distributions. Chapter 2 focuses on always being able to retrieve some items with a given key and spends additional resources for items with rare keys. Chapter 3 preserves privacy by introducing false positives and returning the same number of (hidden) answers for all queries, thus trying to keep the number of answers that are returned for any query constant. Chapter 4 spreads data load evenly across nodes and maximizes the number of items with the same key that are located in the same node.
Chapter 2

Rarity-based routing

A straightforward DHT-based approach for creating a distributed inverted index suffers from a linear increase of messages and replicas with the number of keys. We try to alleviate the problem by proposing a multi-attribute popularity-based routing algorithm which favors rare descriptions over popular ones.

In terms of the goals set in the introduction, this algorithm focuses on scalability and openness, since a large, open number of nodes collaborate to maintain the index. This comes at a price: the popularity-based algorithm forgoes completeness for scalability, as it is not guaranteed to find all answers.

This chapter has been published as “Rarity-based routing in structured overlays” in the third International Workshop on Collaborative Service-oriented P2P Information Systems at IEEE WETICE’07 [64].


abstract

The OpenKnowledge project\(^1\) aims at knowledge sharing through open and flexible peer interactions. Within this project, we are developing a system that supports searching, developing and sharing of interactions/workflows consisting of roles implemented by software that can be shared and executed by peers. Part of this system is a discovery service, which will be the focus of this chapter. This service aspires to fulfill the above requirements featuring a Peer-to-Peer architecture and Distributed Hash Tables (DHTs) to achieve robustness through redundancy and scalability through decentralization. Resources are discovered using a set of attribute-value pairs. A straightforward DHT-based approach that creates a distributed inverted index suffers from a linear increase of messages and replicas with the number of attributes. We try to reduce this number by proposing an efficient multi-attribute routing algorithm. We emulate and test our implementation on the DAS-2 distributed supercomputer.

2.1 Introduction

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

The EU-funded OpenKnowledge project\(^2\) has as one of its goals to build a P2P system, which we call the OK-system, for sharing knowledge, not only in the form of data but also in the way the data is processed \(^3\). Using this system, people can publish workflow descriptions (also called Interaction Models or IMs), and peers can subscribe themselves to play one or more of the roles in them. The role-code (i.e. software) that a peer needs to have to play such a role can also be shared and downloaded via the OK-system. Peers can also subscribe themselves to be coordinators of IMs, being responsible for their correct execution. All components of the OK-system will be implemented in a way that everything runs distributed without any central control. In this chapter we focus on the component responsible for finding:

\(^1\)http://www.openk.org

\(^2\)for additional information about OpenKnowledge, the reader is referred to: http://www.cisa.informatics.ed.ac.uk/OK/deliverables.html
2.1. INTRODUCTION

• **Interaction Models** Interaction models define the way services interact, expressed in a formal language like LCC[98] or BPEL\(^4\). They are described by a set of attribute-value pairs, they have small size and their search is facilitated by multi-attribute search. In table 2.1 an example descriptor of an IM is given. The unique identifier of the IM is `Auction5443FF`, and it is described by a set of attributes pertaining to its use and characteristics. These attributes (e.g. **Type** or **Role**) are fixed for OpenKnowledge, although we will see later that our discovery service does not depend on such a fixed schema.

• **Service descriptions** Services are described by a (potentially large) set of attribute-value pairs. They are small in size, and expected to be transferred over the network often. Consequently, we need efficient mechanisms for multi-attribute search, incorporating methods from information retrieval. In table 2.1 we show an example descriptor of a service description. For example, a user that has found the example interaction model can search for `{′IM=Auction5443FF′,′Role=Buyer′}` to find compatible service implementations for the role she wants to play, which is possible because a service description has at least one pointer to a role it implements for a given IM.

• **Service implementations** Service implementations are pieces of software statically bound to a service description. In the context of the OpenKnowledge system, they are pieces of mobile code that are described by a service description.

• **Coordinators** Finally, we need functionality to find coordinators. As can be seen in table 2.1, a coordinator should specify the languages it is able to interpret. Intelligent as opposed to random selection of coordinators for an interaction can have many advantages (e.g. if we select the same coordinator for multiple invocations of the same interaction, we can add interaction participants at runtime).

To the best of our knowledge, there exist no scalable, efficient and fully-distributed implementations for multi-attribute indexing and search. The JXTA project claims to have implemented such a system, but unfortunately, after conducting extensive tests, we discovered that its implementation could not scale to more than a handful of rendezvous peers[62]. The focus of this chapter is on scalable, open, efficient and robust publishing and discovery of these resources through a community-supported Peer-to-Peer system.

DHT implementations [95, 101, 1, 74, 11] are currently seen as an important building block for Peer-to-Peer systems for storing content in a completely decentralized way [29]. Nodes function autonomously and collectively form a complete and efficient system without any central coordination. In DHT overlays, each object

\(^4\)http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpe
1. Auction5443FF {Type=IM, Descr.Term=Auction, Descr.Term=Buy, Descr.Term=Sell, Descr.Term=Dutch Auction, Role=Buyer, Role=Seller}

2. Buyer4325 {Type=Service description, IM=Auction5443FF, Role=Buyer, Descr.Term=Buy}

3. CoordinatorFF32 {Parser=LCC, Parser=BPEL, Certification=Verisign}

Table 2.1: Examples of resource descriptors.

is associated with a key, chosen from a large space. This space is partitioned in zones, and each peer is responsible for the keys and corresponding objects in a zone. Peers need to maintain connections only to a limited number of other peers and the overlay has the ability to self-organize, with respect to peer connections and object distribution, to handle network churn. In principle, all DHT-based systems provide the following functionality: store(key, object) storing an object identified by its key, and search(key) which returns the object (when it exists) from the peer responsible for the key. Current systems need approximately O(log(N)) messages to search or store and each peer needs to keep from O(1) to O(log(N)) pointers to other peers, where N is the number of peers in the network [74, 95, 101, 1]. Although they seem to deal very well with key lookups, automatic load balancing and robustness, their application domain is constricted by the absence of efficient methods to search for richly described content.

The discovery service of OpenKnowledge is implemented as a layer on top of DHTs to provide efficient discovery, multi-attribute search and distributed storage through use of the semantics of data being stored. In this chapter, we are providing the fundamentals for such a system by presenting a novel popularity-based algorithm for multi-attribute search over richly-described content. The OpenKnowledge system is more extensively described in chapter 9.

In section 2.2 we outline the motivation our research. Our approach is described in section 2.3. We give an evaluation of the system in 2.4 and we conclude our work in section 2.6.

2.2 Multi-attribute search by joining single-attribute searches

It is relatively easy to design a discovery system over a DHT that maintains a distributed inverted index over all attribute-value pairs. Namely, to insert a new resource into the system, one could hash each attribute-value pair and store it together with a pointer to the resource in the DHT overlay. Furthermore, to retrieve
2.2. MULTI-ATTRIBUTE SEARCH BY JOINING SINGLE-ATTRIBUTE SEARCHES

a resource, for each attribute-value pair of a query, a lookup on the hash of this pair is performed and a local join is performed over the results.

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

- **Distributed join is costly** To perform a distributed join, the initiating peer has to gather all index entries for all the attribute-value pairs in the query and perform the join locally. The cost of this can be prohibitively high, especially for queries with many pairs. The problem is further aggravated by their distribution of frequency. In document retrieval systems, terms usually follow a zipf distribution[19] (also see figure ??). Therefore, a query with at least one of these common attribute-value pairs will be too expensive to calculate, since it would imply retrieving all the indexes containing that attribute-value.

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

We can circumvent this problem by storing the entire description to the peers that correspond to the hash of each attribute-value pair. This comes at the cost of additional storage and bandwidth costs for inserting descriptions but it makes query answering a local operation and querying now costs only 1 message in the DHT. In the next two paragraphs, we see why this is still inadequate.

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

2.3 Our approach

As a solution to the problem of managing large descriptions and multi-attribute search, this work is focused on popularity-based approaches. The key idea is that popular content is easily available on the network due to a high degree of replication. Therefore, we do not need to spend much effort on indexing it, in contrast to rare items.

In [116], the authors suggest that for queries for common items, flooding queries is sufficient, while for rare items, DHTs perform best. Research in the context of the PIER project and in [71] also suggests a hybrid flooding/DHT mechanism (albeit with no efficient way to determine which items are rare). Indeed, for commons terms we are not interested in getting all results, if there are millions of them; a hundred would be enough. On the other hand, for rare terms, we are interested in all results. Nevertheless, most popularity-based approaches assume prior knowledge of which items are popular which is unrealistic. Our approach is to use statistical information, which is automatically calculated in a distributed way, to determine, on-the-fly, which terms are rare and which queries refer to them, and adapt the routing process accordingly.

An interesting and relevant approach is Mercury[16], supporting efficient multi-attribute and range search using a small to medium-size schema. It relies on hubs, consisting of a collection of peers responsible for storing indexes with a specific attribute, functioning as a sort of sub-overlays. Descriptions are routed to all hubs, which means that the number of messages and replicas is proportional to the number of attributes in a description. Furthermore, each peer in the network needs to know at least one peer in each hub, meaning that the number of peer references that need to be kept by each peer is at least proportional to the number of attributes in the system. Needless to say, this approach cannot scale beyond a schema with a dozen of attributes.

In this chapter, we focus on index placement and do not investigate caching techniques, or the use of shortcuts to peers that gave good results in the past[115]. The methods proposed in this chapter are orthogonal to them and it may be expected that both can benefit from each-other.

The novelty of our approach lies in exploiting the structure in DHTs to extract statistical information useful for routing, with the goal of alleviating the problems

5http://pier.cs.berkeley.edu
2.3. OUR APPROACH

Algorithm 1 Rarity-based walk

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

Ensure: \(d\) is stored.

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

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 return
2: else
3: \(t := t_m \in (t_1 \cdots t_n) | \forall t, |Dt| > |Dt_m|\) and \(P_t \notin Pr\)
4: if \((t = \emptyset)\) then return
5: else
6: \(Pr := Pr \cup this\)
7: send\((q,P_t_m,Pr)\)
8: end if
9: end if

with joining single-attribute searches and current popularity-based approaches. We will describe an algorithm that uses statistical information from the local storages of peers to place descriptors more efficiently. The intuition behind our popularity-based approach is that rare attribute-value pairs are preferred for replication, since:

1. For common attributes-values, it is likely that we will find answers anyway, since more matching descriptors will exist in the system.
2. Rare attributes-values yield a higher information value.
3. Peers responsible for common descriptions are likely to be overwhelmed by descriptions.

To illustrate our case, imagine the following resource description: In the simple approach described in 2.2, the description ‘Buyer4325’ would be replicated to the peers responsible for ‘Type=Service description’, ‘IM=Auction5443FF’, ‘Role=Buyer’ and ‘Descr.Term=Buy’. It is reasonable to expect that a query for \{‘Type=Service description’\} would be easily satisfied by many peers. Therefore, it would be a waste of resources and a network hot-spot to replicate the description to the peer responsible for this attribute/value pair (i.e. to the peer where the string ‘Type=Service description’ maps to). On the other hand, ‘IM=Auction5443FF’ is rare, and the description should be replicated to the peer responsible for it, since it would be difficult to find another peer that has this rare attribute-value. But how do we determine whether an attribute-value is rare? Unlike previous approaches, we do not assume external or centralized sources of statistical information, but rely on the
properties of the distribution of terms into descriptions and the DHT. So, going back to our example, assume that initially (by random choice) the peer responsible for ‘IM=Auction5443FF’ is selected for replication. It is very likely that this peer will already have descriptions with ‘Type=Service description’ since (a) ‘Type=Service description’ is a very common attribute-value and (b) ‘IM=Auction5443FF’ and ‘Type=Service description’ are semantically correlated. Therefore, in our approach, it should decide not to replicate it to the peer responsible for ‘Type=Service description’ since subsequent queries including ‘IM=Auction5443FF’ would be answered by the peer responsible for ‘IM=Auction5443FF’, while queries including for ‘Type=Service description’ can be easily answered by many other peers (or at least by the peer responsible for ‘Type=Service description’).

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

**Inserting descriptions** Insertion messages consist of the description and the attribute/value pairs of the description that have already been used (this set is initially empty). All attribute/value pairs with frequency over a given threshold system parameter $D_{tm}$, over the descriptions of the local peer are marked as used. The attribute/value pair that has the lowest frequency and has not been used is selected (i.e the attribute/value pair with the smallest number of occurrences in the descriptions stored locally in the peer). If such an attribute-value pair exists, it is marked as used and the message is forwarded to the peer responsible for that attribute/value pair in the DHT.

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

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

Furthermore, it is interesting to note its anytime behavior. In the beginning, when peers have no overview of which terms are rare, they will replicate descriptions to all peers, since the threshold for replication will not be reached. As the number of descriptions in the system grows, so will the local knowledge in each peer about which attribute-values are popular, since peers can approximate the popularity of an attribute-value by counting the occurrences in their own data. No additional mechanisms to decide whether there is enough information are required. If a certain attribute-value pair never becomes popular, the system will behave like a DHT, for that pair.
2.4 Evaluation

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

2.4.1 Dataset

Since, to the best of our knowledge, there exists no large dataset for resource descriptions for service workflows/interaction models, we decided to use a dataset created for general-purpose information retrieval, developed for [111]. We believe that this is a realistic assumption because, in a discovery setting attribute-value pairs in a description are semantically correlated. Future research should give additional insight on attribute-value distribution. For now, we assume that it is similar which the distribution of terms in documents. Our dataset was created by crawling a large number of real user queries from SearchSpy⁶ and applying a natural language processing method on the results retrieved for these queries using Google⁷, to get relevant descriptions. The input to our system was derived from the following:

- **Corpus** We have used a corpus of 260,000 documents, resulting in the same number of descriptions. Each description consists of a set of terms.

- **Descriptions** From these descriptions, we have selected a random set of 100,000. On average, each document contained approx. 104 terms (the distribution is shown in fig.2.1). The distribution of terms, as expected, follows a zipf-like distribution(fig.2.1). It is interesting 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: (a) Randomly pick the number of terms $|t|$ for each query (according to the distribution in fig.2.1(bottom right)). (b) Pick at random a description out of the corpus. (c) Pick $|t|$ terms (randomly using a uniform distribution) from the chosen description and use them as the query terms. For most queries, there are fewer than 50 answers.

2.4.2 Criteria

In this chapter, we will evaluate our system in terms of description recall. To gain additional insight, we will always take into consideration the number of answers in the system, but limit the number of answers we are interested in (e.g. maximum 50 answers, which is a parameter that can be changed for each query). The reason

⁶http://www.infospace.com/info.xcite/searchspy
⁷http://www.google.com
behind this, is that, in a discovery setting, there is no point in trying to retrieve all answers. Instead, we are interested in getting enough answers to satisfy the user. Nevertheless, this is not a limitation of our algorithm itself, it is a choice to make our evaluation more realistic. Thus, for our experiments, recall is defined as follows:

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

2.4.3 Design, implementation and experimentation

Design Our system is based on a three-layer architecture, the bottom layer consisting of a DHT implementation. The second layer consists of an object store and a distributed index supporting multi-attribute search and relies on the algorithms described in 2.3. Finally, the third layer is application specific, in our case the OpenKnowledge service and peer discovery.
2.5. RESULTS AND DISCUSSION

**Implementation** We have implemented our system using Java 1.5. For the bottom layer, we have used the FreePastry DHT implementation, version 2.0b\(^8\). The second layer is an implementation of the algorithm in section 2.3 and the subset replication approach. The application on top is the discovery service of the Open-Knowledge system\(^{30}\), as described in the introduction of this chapter.

**Testing and experimentation** We have used the DAS-2 distributed supercomputer\(^9\) to test and evaluate our system. One node on the DAS-2 acted as a bootstrap, being used as an access point to the system for the rest. We have used Globus\(^{10}\) to start 500 instances of our system, which contacted the bootstrap node, and self-organized according to the Pastry protocol\(^{101}\). This process took less than 5 minutes. Next, nodes published in parallel 200 descriptions each (100,000 in total). Finally, each node made 100 queries and collected the results.

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

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

### 2.5 Results and discussion

Figure 2.2 shows the results for the two approaches. Each approach required the same number of query messages, an average of 2.5 DHT messages. The first impression is that, for both approaches and for queries with more than 300 matching descriptions, we get almost perfect recall using ten times less replicas and messages compared to an approach that would replicate according to all attributes/values. Our rarity-based walk yields a recall in excess of 60% for queries with only a single matching description, which are the most difficult to answer, an increase of approx. 35%, compared to the subset replication approach. In total, even for a relatively small overlay of 500 peers, we gain a substantial increase in recall using the same number of messages. It is expected that as the network size grows, the recall of the

---

\(^8\)http://www.freepastry.org  
\(^9\)http://www.cs.vu.nl/das2/  
\(^{10}\)http://www.globus.org/
subset replication will deteriorate faster than that of our rarity-based walk approach, since the load balancing problems of the former will be augmented. In Figure 2.2, a drop in recall can be observed for queries with 3-50 matching documents. We defer explanation for this drop to Chapter 5.

2.6 Conclusions

In this chapter we outline the functionality and design of a scalable peer-to-peer discovery service. We provide an implementation that incorporates a novel algorithm that reduces the scalability problems of multi-attribute indexing and search in DHT networks by automatically calculating attribute/value popularity and using it to reduce the degree of replication. Our implementation was tested by emulating 500 instances of our system on the DAS-2 supercomputer. The results indicate that an algorithm that takes attribute/value popularity into consideration for routing outperforms an algorithm that does not. Future work lies in testing how this gain in performance changes as the network size increases and measuring the robustness of our system toward a high peer churn rate.
Chapter 3

Discovery of private resources

In an open environment and on an untrusted index, it may be required to protect individual entries. We propose a method to collaboratively maintain a distributed index without exposing terms of concrete mappings.

With regard to the goals set in the introduction, this method focuses on scalability (in terms of the size of the index), control (since nodes do not have to make the descriptions of their information public and maintain complete control over their data) and openness (since the participants in the index do not need to be trusted, thus any node may join) and consistency (since participants maintain their own data).

This chapter is the article “Scalable discovery of private resources” published in the third International Workshop on the Value of Security through Collaboration at IEEE SECURECOMM’07 [65].
abstract

Resource discovery is fundamental to a multitude of distributed systems, including grids, web-based applications and multi-agent systems. To achieve scalability at a low cost, many researchers have turned to a peer-to-peer paradigm, leading to the development of a multitude of protocols and algorithms being developed, with implementations still lagging behind. In this chapter, we consider the privacy implications of peer-to-peer discovery systems and propose a framework for discovery of private resources. Furthermore, we propose and evaluate an architecture and a series of methods using distributed hash tables. Finally, we provide an implementation in the context of the OpenKnowledge project.

3.1 Introduction

Resource discovery is about localizing computational resources. It is of paramount importance for a large proportion of computer systems. Practically all dynamic distributed systems require mechanisms to enable users and applications to locate resources scattered across the network. Notable examples are Grids, file and knowledge sharing systems, peer-to-peer systems, multi-agent systems and distributed knowledge bases. Despite their incongruence in architecture and assumptions, all such systems require at least this basic functionality: retrieving the location of resources given a set of keywords or attribute-value pairs.

With the notable exception of peer-to-peer search engines [58], most currently deployed systems are based on centralized architectures (e.g. typical search engines and UDDI). Nevertheless, such architectures are typically associated with a number of flaws: there is a single point of failure or a single point of administration, in turn leading to lack of scalability and selective or preferential disclosure of information, censorship, preferential ranking of results and centralization of information respectively.

Peer-to-peer systems are often seen as a vehicle for the democratization of distributed computing, since a collection of community-volunteered peers standing on an equal footing, collaborate to achieve a common goal. This architecture is perceived as capable of alleviating most of the problems mentioned above. Nevertheless, as we will show, current peer-to-peer discovery systems not only fail to solve privacy infringement issues, but they are actually more vulnerable than centralized approaches. The main reason behind this is that instead of the existence of one central point where privacy can be infringed, any peer in the system is a potential
3.2. PRIVACY IN PEER-TO-PEER DISCOVERY SYSTEMS

point of attack.

Furthermore, we argue that the techniques that are traditionally employed to provide security over unreliable overlays are inadequate as far as protection of privacy is concerned. For instance, although a quorum-based approach relying on a number of peers voting to perform an action can be used to bestow robustness against malicious peers [47], it is detrimental in the event that this action has to remain private.

The focus of this chapter is on scalable discovery of private resources using private queries. The term private signifies that peers, other than the one posing the query and the one answering, should not be able to view either the resources published or the queries posed (or a meaningful description of them). The term scalable signifies that it should retain good performance as the number of peers, resources and concurrent queries increases.

Initially, we examine how current solutions fare with privacy and define a framework for discovery of private resources. Based on this framework, we develop an obfuscated index on top of a Distributed Hash Table (DHT), where resources are published without revealing either their location or their description. In addition, we implement this method as part of the OpenKnowledge discovery [31].

3.2 Privacy in peer-to-peer discovery systems

The purpose of discovery systems is to provide location services for large pools of resources. To this end, users or applications publish resources by sending a descriptor, an identifier and the location of a resource to the discovery system, and query, which means searching for the location of resources given a set of criteria. Once the location is established (i.e. when the discovery system has returned the locations of the resources which have descriptors matching the query), the querier may proceed to negotiate with the publisher of the resource.

Using the above terminology, we can identify a series of privacy challenges concerning the disclosure of the following information:

- **Identity of the publisher** Information that can identify which user or through which host a particular piece of information was published. For example, the IP address of the host, along with an identifier to the resource.

- **Identity of the querier** Information that can identify the issuer of a query.

- **Resource descriptor** A meaningful descriptor associated with an identifier of the resource.

- **Resource location** The location where a resource resides, or an association of a resource identifier with its location.

- **Content of the query** The content of a query, regardless of whether it is associated with a querier.
We advocate that the peer-to-peer paradigm can be used to preserve privacy as well as provide scalability and robustness in discovery systems.

We urge the reader not to confuse anonymizing networks with peer-to-peer discovery. The former refers to providing communication using anonymous endpoints which are known in advance while the latter refers to providing discovery functionality through a peer-to-peer network. Although it should be possible to provide an integrated solution, in this chapter we consider the two problems orthogonal.

The benefits of peer-to-peer architectures for discovery have been extensively studied [127]. Nevertheless, current peer-to-peer discovery systems do not fare well with privacy. In the following paragraphs, we will provide an overview of the privacy implications of current peer-to-peer systems in combination with their scalability properties organized according to overlay type.

### 3.2.1 Centralized index

The simplest scalable solution is to have a central register of keys that maps to the peers that store the resources. Napster [79] was such an example where resource descriptors and their locations are stored in a central register. For a query the register returns the location of the peer(s) that hold the matching resources. Although it was scalable in storage due to content distribution, there is a single point of failure which makes this architecture not robust against failure of the index node.

### 3.2.2 Unstructured overlays

In unstructured overlays, peers maintain a set of ad-hoc connections to other peers, also termed as *neighbors*. According to the criteria used to establish and preserve these connections, a further distinction can be made between network-based and semantic-based unstructured overlays. The former use information concerning the network to self-organize on, for example, hop count and latency. The latter use descriptions of the content of each peer, for instance, summaries of filenames or sets of descriptive keywords.

**Network information-based overlays** The first fully-distributed peer-to-peer networks [58] use information concerning the underlying IP network to self-organize. Typical criteria to establish neighbor relations include number of IP hops and link quality.

Searches are typically flooded through the network, i.e. sent to all neighbors and forwarded further, similar to a breadth-first search. Although a series of improvements over the original model have been proposed, like focusing on reducing duplicate messages [133], each query still has to be propagated to a large percentage of all peers, making this approach non-scalable for resource discovery.
As far as privacy is concerned, the main advantage of this category of systems is that peers do not need to share either their resource descriptions or their location. Nevertheless, the content of queries and the querier are public.

**Semantic-based overlays** Semantic-based overlays are similar to network information-based unstructured overlays but for the fact that they use semantic information about their resources instead of information about the network. In most cases, peers propagate summaries or descriptions of their data, and cluster according to content similarity or positive past experience.

Queries are propagated either in a similar fashion as with network-based overlays, or using similarity measures. In some cases, semantic-based overlays have shown performance improvement of one order of magnitude compared to network-based approaches, measured in terms of recall [113].

Unfortunately, this comes at a cost of reduced privacy as content descriptions need to be shared in order to perform peer clustering effectively. On the bright side, peers can choose to whom their descriptions will be sent (i.e. they may send descriptions of their content only to peers that they trust).

### 3.2.3 Structured overlays

Structured overlays impose a global structure on the peer-to-peer network. DHTs is a fecund research field with a plethora of applications [95, 101, 1, 74, 11]. Typically, each item stored in the DHT is associated with a hash ID chosen from a large key space. This space is partitioned, in a way similar to hash tables, but instead of bins, they have peers. This distributed data-structure is self-maintaining, self-organizing and guarantees lookups and insertions using, in most cases, \( O(\log(N)) \) messages, where \( N \) is the number of peers in the DHT overlay. Each peer is required to maintain a number of connections\(^1\) to other peers according to the structure of the DHT. Therefore, their scalability properties are much better compared unstructured overlays, albeit with increased maintenance costs and reduced robustness.

To discover resources using a single attribute, the overlay can act as a distributed index, mapping from keywords to locations or identifiers. Structured overlays need only a single peer lookup. Furthermore, some of them also support range queries [1, 74]. For multi-attribute search, more complex approaches have been developed. Other systems place data according to their semantics. For example, pSearch [49] uses Latent Semantic Indexing to map documents in a multi-dimensional real-valued space maintained by a CAN.

Privacy-wise, current discovery systems based on structured overlays have the worst characteristics. Firstly, resource descriptions have to be forwarded and indexed by the peer determined by the hash function used in the system. Not only does this not guarantee privacy, but it is also a potential security threat since peer placements

\(^1\)logarithmic to the number of peers in the network [95, 101] or constant [75]
in most overlays are not secure. This enables peers to ‘select’ the keywords they want to index, making targeting of specific topics easy. This means that the identity of the publisher, the content description and the resource location are disclosed to all peers routing the respective messages and to the peer(s) indexing the resource description.

Secondly, queries are propagated by a set of peers determined by the DHT mechanisms, again, not guaranteeing the privacy of the querier or the query content. Similarly, peers may choose to be responsible for the propagation of the queries they are interested in, further compromising privacy.

In short, structured overlays and systems based on them clearly outperform unstructured ones as far as scalability is concerned. Nonetheless, privacy has been taken in little or no consideration in their design. In the next section, we will define a framework, outlining where privacy attacks can take place, followed by an implementation of a mechanism to improve the privacy properties of peer-to-peer discovery systems.

### 3.3 A framework for discovery of private resources

As shown in the previous section, disseminating information about peer contents either in the form of advertisements, as in semantic unstructured peer-to-peer overlays, or in the form of a distributed index, as in structured overlays, alleviates the scalability issues of peer-to-peer discovery systems to a great extent. Nonetheless, it gives rise to significant privacy issues. In this section, we will provide a framework for the design of a peer-to-peer discovery system, where resource descriptions will be published in a global index, and fine-grained matching and negotiation will be done on peer level.

In figure 3.1, one can see an overview of the proposed model. Publishers send a descriptor of the resource they want to share along with the endpoint where they want to be contacted, for example their IP address and port number. This information is stored in a global index. A querier can send a query to this index, specifying
3.4. A DISTRIBUTED OBFUSCATED INDEX

A descriptor for the resource to be matched. Assuming a matching descriptor is stored in the index, the endpoint of the peer that has published it is returned to the querier. The latter contacts the publisher directly and they negotiate further, disclosing or requesting additional information about the requested and offered resource and participating parties.

The above model will be used to elicit additional scalability and privacy requirements. Concerning the former, it is obvious that the global index is the performance bottleneck of the system, as all other operations scale linearly with the number of peers. Therefore, a scalable global index is required(1). As far as privacy is concerned, the issue becomes slightly more complicated: Firstly, we cannot assume a trusted global index, since that would hamper scalability. Therefore, published descriptors should convey as little information as possible. Secondly, these descriptions should preferably not be understandable by the index(2). Thirdly, queries should also not be easily understandable by the index(3). Fourthly, it should not be easy to probe for peers that published or queried for a particular type of resource, i.e. it should not be easy to identify the peers offering resources with a given description(4).

3.4 A distributed obfuscated index

Following the requirements above, we propose a scalable peer-to-peer discovery system based on a distributed obfuscated index, on top of a DHT. Transposed to a real world scenario, our model is similar to the following: Arnold leaves a note on a public message board:

- I have some ho.... Call me at 234435

Barbie is searching for somebody that can give her some “hot...”. She goes to the message board and checks for messages that match hers. She can see that Arnold’s message matches hers, so she calls Arnold and says:

- I am Barbie and these are my credentials. I saw that you might have something for me. I am searching for something starting with “hot...”

Arnold responds giving his credentials. Barbie considers that he is trustworthy and tells him that she is searching for “hot soup”. Arnold responds that they do not match (without disclosing any additional information about his message).

Obviously, this message board would become crowded very fast. We need a more efficient way to find interesting notes. Assume that there is a message board on the door of each room in a corridor. These doors are numbered in sequence from 1 to 100. To determine where each message should be posted, assume the existence of a function that maps words to numbers in the range [1, 100] and that similar words are mapped to similar values. For example, say that Arnold’s note “ho” maps to 5.4 and that Barbie’s query “hot” maps to 6.1.

Furthermore, assume that Maria, a student in the university, has to post the notes behind the door with number that lies closest to the number of the word, (i.e. a word that maps to 32.45 will be posted behind room 32).
In the example above, Arnold’s note would be posted behind door number 5 (5, 4 is rounded to 5).

Maria will map the query of Barbie to room number 6. Assuming that no matching note will be found there, she will search in the message boards of nearby rooms. In room 5, Maria will find Arnold’s note reading “5.4, Arnold@653435”. She will return a copy of the note to Barbie. Barbie will then contact Arnold in person, and they will negotiate further.

It is important to note the following:

- It is difficult to find out the content of the message from its number/hash. Furthermore, in order to improve security, we can round numbers. For example, assume that “Hot s” maps to 6,234. If we round it to 6, it is much more difficult to guess what word or phrase is meant, since there are many words that map to 6. Nevertheless, that would come at the expense of searching in more rooms.

- Persons in the rooms may be malicious and want to spy on the information from the notes. This is not possible, since they cannot infer their meaning from their number to which they were mapped.

- Students gossip, so Maria may want to spy on information in the notes. Again it is not possible, since Maria cannot infer either what Arnold’s query means, or Barbie’s note, for the same reason as mentioned before.

Mapping the terminology above to computer science terms is as follows: The number of a word is its hash-value using a similarity-preserving hash-function, a corridor of rooms with message boards is a DHT, the student Maria is the message routing algorithm, a message board is a node in the DHT and rounding numbers is comparable to truncating hash-values (i.e. using a prefix of the hash-value).

Figure 3.2 outlines this functionality of a system based on an obfuscated index. We are assuming that resource descriptions and queries are made up by sets of
3.4. A DISTRIBUTED OBFUSCATED INDEX

keywords (e.g. “country”, “car”). To publish a resource descriptor, peers replace each keyword in that descriptor with a prefix of its hash, denoted as $p_i(h)$, where $i$ is the length of the prefix and $h$ the hashed keyword. This obfuscated descriptor is stored in the distributed index (maintained by the DHT), therefore DHT nodes do not have access to the original descriptors. Similarly, queriers replace each keyword in the query with its hash. Again, the original queries are not visible by the nodes in the DHT. The distributed index will return the endpoints of the peers which have registered descriptors with the same prefix as the query.

The role of $i$ in the hash function acts as a knob to adjust the trade-off between privacy and performance. A high $i$ means that few keywords will map to the same hash-value. This means that precision, which is the chance that the returned peers actually contain relevant resources, will be high. Nevertheless, the system would be susceptible to dictionary attacks, i.e. malicious peers trying to guess descriptors. On the other hand, a small $i$ would increase the number of collisions in the hashes. This would make it impossible for attackers to make guesses about the content of a descriptor, since multiple uncorrelated keywords will map to the same hash-value. Of course, this will come at the expense of additional negotiation steps (marked with 4 in fig. 3.2). Note that for $i = 0$, the system would degrade into a broadcast-based system where all peers would be returned by the index while for a very high $i$, the obfuscated index would behave as a distributed inverted index.

It should also be noted that $i$ can be set per keyword and per peer basis. Therefore, for public descriptors and queries, a high $i$ ensures scalability, while for private ones, a lower $i$ ensures that it is very difficult to guess their content.

In the next section we provide a method how to determine $i$ for individual keywords. In algorithm 1, we formally describe how the process of storing pointers in a privacy preserving way. We assume that a document or any other resource $o$ is described by a set of keywords, which we give the symbol $D_o$. For each keyword in $D_o$ we calculate the key $p_t$ using one of the following options:

- **Option 1: Secure hash** Calculate the secure hash of the keyword and take a prefix of length $l$ of this hash.

- **Option 2: Fixed prefix length** For each keyword, take a prefix $l$ equal to the length of the keyword multiplied by ratio $r < 1$. The higher the $r$, the longer the prefix would be.

- **Option 3: Determine the prefix of the keyword according to its information value** This method is further described in section 3.5.2.

Then, the descriptor of the resource, along with a pointer to its location should be routed based on the key $p_t$ calculated above. The way that this key is routed is DHT implementation specific.

In algorithm 2 we formally describe how resource discovery is done. The functions to calculate the information value, the prefix, the key and the routing are identical.
to storing. Then the DHT is used to retrieve the locations of the resources matching these keys. Finally, the querier negotiates with each of the returned peers.

Algorithm 2 Privacy preserving storage

Let resource descriptor $D_o := \{t_1, t_2, ..., t_n\}$, be a set of keywords, where each $t \in STRING$ is a keyword describing the resource to be published.

for $i := 1$ to $i := n$ do

[OPTION 1: secure hash]

$p_t \leftarrow \text{secure}\_\text{hashkey}(t_i, l)$ // $\text{secure}\_\text{hashkey}$ is a function that calculates a secure hash key, e.g. via SHA-1 into the key space of the DHT.

[OPTION 2: fixed prefix length]

$p_t \leftarrow \text{prefix}(t_i, r)$ // $\text{prefix}(t, l)$ is a function that calculates the prefix of string $t$ with length $t.length \ast r$.

[OPTION 3: Information value-based prefix length]

$i_t \leftarrow \text{infValue}(t_i, Dic, d)$ // $\text{infValue}$ is a function that calculates the information value $i_t$ of the keyword $t_i$ according to a dictionary $Dic$. This function will be explained in the next section.

$p_t \leftarrow \text{prefix}(t_i, i_t)$

dhtRoute($p_t, addr$) // $\text{dhtRoute}$ routes the location of the resource ($addr$) via the DHT with key $k_t$

end for

3.5 Design choices

In this section, we further describe some important design choices and discuss their implications.

3.5.1 Mapping function

DHTs work within an key space, which may be implementation-specific. For example, Pastry uses a single dimensional space with identifiers of 160 bits whereas CAN uses a multi-dimensional, real-valued cartesian space. Therefore, it is necessary to map keywords in the key space maintained by the DHT. Trivial as it may seem, it gives rise to load-balancing problems and has privacy and performance implications. In this section, we will give an overview of three ways to perform this mapping and discuss their characteristics.

No hash

Keywords are stored in the DHT in their original representation, possibly transcoding them to match the representation in of the DHT. Apart from simplicity, the
Algorithm 3 Privacy preserving retrieval

Let $\alpha$ be a system parameter indicating the maximum number of peers to start a negotiation with, once they are found via the algorithm.

Let query descriptor $Q := \{t_1, t_2, ..., t_m\}$, be a set of keywords, where each $t \in STRING$ is a keyword in the search query.

Let $A := \emptyset$ be an empty set of $(addr, rank)$ tuples containing peer addresses $addr$ with a ranking $rank$ indicating the number of prefixes it shares with the query $Q$.

for $i := 1$ to $i := m$ do

[OPTION 1: secure hash]

$k_t \leftarrow secure\_hashkey(t_i, l)$

[OPTION 2: fixed prefix length]

$k_t \leftarrow prefix(t_i, r)$

[OPTION 3: Information value-based prefix length]

$i_t \leftarrow infValue(t_i, Dic, d)$

$k_t \leftarrow prefix(t_i, i_t)$

end for

$A = dht\_Retrieve(k_1...m) // retrieves a, possible empty, set of addresses via the DHT given the keys $k_1...m$ and stores it in the variable A$

start\_Negotiations(A, \alpha) // starts direct negotiation with the $\alpha$ best ranked peers

The most important advantage of this approach is that it is likely that keyword similarity is preserved. For instance ‘computer’ and ‘computers’ will have very similar keys and, most likely, will be stored in the same peer, providing good data locality. Nevertheless, privacy becomes an issue, since peers in the DHT will be able to see resource descriptions.

To make matters worse, keys will be unevenly distributed in the key space of the DHT. To illustrate our case, in English there are much more words starting with a ‘t’ than with an ‘x’. This would limit the choice of DHTs to ones that support non uniform distribution of keys(e.g. [1, 74]).

Secure hash

An alternative would be to use a secure hash (e.g. SHA-1) and store the hash-values of keywords in a description. Such functions have the property that they map values uniformly in the key space, which means that descriptions are evenly spread across peers, leading to good load-balancing properties. Furthermore, they are one-way functions and it is very difficult to find a value that hashes to a specific key. This property is desirable to make sure that descriptors are kept private. Nevertheless, a dictionary-like attack is still possible. For example, an attacker participating in the DHT may calculate the hash-values of a large number of keywords from a dictionary and try to infer the original values in the descriptions it is storing. This is the reason
that it is still required to use only a prefix of the hash, instead of all of it (also see next section). Finally, secure hashes do not preserve similarity between keywords: for example, ‘coffee’ and ‘coffees’ would map to completely different keys. This makes it impossible to support approximate matches, like edit distance matches or \textit{startswith} matches, in the case it would have otherwise been possible.

\textbf{Similarity-preserving hash}

A plethora of similarity-preserving hash functions have been proposed [56] for a variety of applications. To the best of our knowledge, there are no secure or load-balanced variants of such functions. Therefore, the challenges presented are the same as when using no hash-function at all.

\subsection{Determining key length}

As previously mentioned, we assume that resources are described by a set of keywords $t_1, t_2, \ldots, t_n$.\footnote{Note that \textit{keyword} should be interpreted in the broadest sense, as a descriptive string which could be meaningful for an application but meaningless in natural language. E.g. “ID43XT” or “\texttt{cntry\_code=NL}”} Please recall that the goal of having a prefix instead of the whole keyword is that the shorter the prefix the more other resources match, which makes it more difficult to know what the resource of interest was, and therefore increases the privacy of the querier and publisher. In other words, when a peer wants to determine the length of the prefix from a key for an resource, the peer should have the guarantee that, in most cases, this key maps to more than X other resources stored in the network. The bigger the X, the larger the privacy, but also the larger the network load. Thus, this X is an indication of privacy and network load. This means, the larger the desired X, the shorter the key needs to be. A key of zero length means retrieving all resources. The variance of the desired average is of influence to the choice on X. For example, (an extreme case) if in 99\% of the times a key of length K maps to one keyword and in 1\% of the time to 901, the average is 10 resources. In this extreme case, in 99\% of the cases K is a unique identifier for the resource and therefore 99\% of the cases completely intrudes the privacy of the user which would be unacceptable both in keywords of privacy and large (but rare) network usage peaks.

We want to keep this variance in number of matching descriptions to a minimum. Imagine that the keys would be prefixes of words from the English dictionary. The popularity of characters is not evenly distributed and neither are their combinations. For example, a key length of three would give sufficient privacy to store the word “computer”, because many words start with “com” (e.g. “compatible” and “communication”). However, it also potentially generates quite some network load because the identifiers of these peers will all be communicated to the querying peer resulting in either large messages or many messages or both, depending to the kind
of DHT used. On the contrary, if the keyword is “xantype”, the key “xan” would be a strong indication that you are storing “xantype” because not many words start with “xan”.

Combined with the type of mappings mentioned in the previous section, we can identify three possible methods to determine key length:

**No mapping and fixed length keys**

For all keywords, the same prefix length is used. For instance, we define that the prefix length is equal to 40% of the number of characters in the keyword. The obvious advantage of this approach is its simplicity. Secondly, assuming that \( i^\beta \), where \( \beta \) is the base of character representation, is much larger than the number of peers in the DHT, there is no need for the DHT to support range queries. This would mean that the key space is large enough so as to be partitioned by the peers in the DHT. The disadvantage of this approach, as we will see in the evaluation, is that the variance is large.

**Key length automatically determined using the prefix information value**

In this method the length of the prefixes for each individual are based on its information value, i.e. popularity. The more popular the prefix, the more keywords match to it, the lower the information value, the higher the privacy. Using a dictionary, one can easily find out how popular a prefix is by simply counting the number of words starting with it. We now assume that the distribution of prefix popularity in a dictionary is identical with the keywords used in the network to describe the resources.

If a dictionary is not present, it is easy to make one by counting the prefix frequency of a set keywords representative of the keyword distribution in the system. A data structure suitable to store this information is a trie. Figure 3.3 shows an example. Such a data structure can be pruned to substantially reduce storage space.

**Using a secure hash**

Secure hash-functions map values uniformly to the key space. Therefore, no extra measures need to be taken to reduce variance. The only challenge is to find an appropriate key length.

### 3.6 Evaluation

The goal of this section is to evaluate the three typical options described above: Fixed ratio size prefixes, information value-based prefixes and secure hashing. We will not evaluate the performance of DHT’s, because it has already been extensively studied [95, 101, 1, 74, 11]. We have two objectives:
Figure 3.3: A Trie data-structure is a tree of arrays containing counters. Each position in each array corresponds to a character. Following the trie one can derive the frequency of prefixes by starting at the root and recursively following the pointers for the current character. For example, the frequency of “A” is 3, and the frequency of “1@” is 4.

- **Influence of the parameters on the balance between privacy and performance.** As mentioned before, the desired privacy influences the performance of the system. All of the options presented before have one parameter that adjust the trade-off between privacy and performance. We perform a series of experiments to assist future implementors in choosing the right values for their applications.

- **Variance on the number of returned peers and its impact on the privacy and performance.** To preserve the privacy of the querier and publisher, a minimum number of peers need to be returned that do not match the original query but only the prefix. On the other hand, if this number is too big, it will have an adverse effect on the performance. Clearly, the variance in the number of peers returned should be as low as possible because a high variance may result in compromising privacy and/or unacceptable performance loss. We use this criterion to evaluate our three options.

### 3.6.1 Dataset

Since, to the best of our knowledge, there exists no large dataset for resource descriptions, we decided to use a dataset created for general-purpose information retrieval, developed for [111]. Due to the semantic correlation between the keywords we assume that this dataset is an adequate substitute. Future research should give additional insight on several “keyword” distributions. For now, we assume that it is similar which the distribution of keywords in documents. Our dataset was created by crawling a large number of real user queries from SearchSpy\(^3\) and applying a

\(^3\)http://www.infospace.com/info.xcite/searchspy
natural language processing method on the results retrieved for these queries using Google to get relevant descriptions. The input to our system was derived from the following:

- **Corpus** We have used a corpus of 260,000 documents, resulting in the same number of descriptions. Each description consists of a set of keywords.

- **Descriptions** From these descriptions, we have selected a random set of 75,000. On average, each document contained approx. 104 keywords (the distribution is shown in fig.3.4). The distribution of keywords, as expected,
follows a zipf-like distribution (fig. 3.4). It is interesting that more than half of all keywords appear only 1 time, while 1 keyword appears in more than half of the descriptions.

- **Keywords to make the trie** We split the set of descriptions into 25K and 50K, where the keywords in the former set is used to make the trie. As shown in fig. 3.5(e), for the settings shown below, the trie contains less than 9000 nodes. In this scenario, each node consists of 16 integers along with 16 pointers to other nodes, which totals to 128 bytes per node. Without any compression, the total storage required for this trie would be less than 15MB.

- **Keywords to evaluate the methods** The keywords in the remainder description set (50K descriptions) is used to evaluate our methods, which are in total 320K keywords.

### 3.6.2 Results

We performed a series of experiments to gain insight into interesting values for the parameters from our three different options.

In figure 3.5, we show how many keywords map to each key, for the ‘secure hash’ (fig. 5a), the fixed length prefix (fig. 5b) and the information-based prefix (fig. 5c). So, in the X-axis we represent the keys and on the Y-axis the number keywords that map to each of these keys. In the legends we describe the values chosen for the parameters to adjust the privacy/performance trade-off and the used average number of keywords per key. For each key, a low number of mapping keywords indicates that the privacy for these keywords is not guaranteed. On the other hand, a very high number of mapping keywords indicates that many peers need to be contacted, resulting in low performance. So, ideally, the graphs should be a horizontal line, meaning that for all keys there will be enough mapping keywords to guarantee privacy and few enough keywords to keep performance acceptable. Furthermore, the horizontal line should be vertically adjusted for the desired privacy level. For example, with a horizontal line at Y=10, for all keys there are 10 mapping keywords.

In figure 5a, we can see that the secure hash approach is closest to the ideal case, the horizontal line. For different key-lengths, one can find the corresponding number of mapping keywords. For example, for a 48bits key, on average 78.22 keywords map for the key and the minimum is 50. In figure 5b we can see that the fixed-length prefix approach has both performance and privacy issues. For example, if 33% of the key length is used it could be that the average of 10.39% mapping keywords is fine, however in +/- 50 cases a key maps to more than 1000 keywords resulting in low performance. Even more important, in most cases it matches to only one keyword which means that privacy is diminished for that corresponding key. Figure 5c shows the results of the information value-based prefix approach. By only looking at the slope of the curve we can already see that the results are much better than the approach with the fixed prefix length. In case of an average of 10.29 mapping
3.6. EVALUATION

Figure 3.5: a) **option 1: secure hash.** The number of mapping keywords for each secure hash key (in our case SHA-1), for different prefix sizes. The lines are very horizontal indicating a small difference between the min. and max. number of keywords mapping to the different keys. The legend should be read as follows: a setting with a 48 bits key, where on average 78.2 keywords map to a key. 

b) **option 2: fixed-length prefix.** The results for the option with fixed ratio length prefixes, i.e. 15%, 33% and 40% of the keywords. The slope is far from horizontal and also many keys map to only one keyword resulting in privacy infringement.  

c) **option 3: inf-based prefix.** The results are shown for three different values of d (=min. freq. of the prefix in the dictionary). Compared to b) the slope is much more horizontal, with less keys having only few mapping keywords. 

d) **keywords per key variance.** This figure shows the variance on mapping keywords for each average number of keywords/key , where the latter are reached by adjusting the parameters of the corresponding option (fixed length, secure hash and information based). 

e) **number of trie nodes for different freq. thresholds.** This figure shows how many nodes are needed in the trie to store the keywords, where d is the min. number prefixes should occur in the dataset before a node is created for it. 

f) **peer precision per query.** For each of the three options, it is possible to get almost the same query distribution by tuning the right parameter.
keywords per key, the maximum number is 206 keywords and more than 98% of the keys map to more than one keyword. Figure 5d shows the variance of number of mapping keywords per key. As expected, the secure hash approach has the least variance. The information-based prefix approach is an order of magnitude better than the fixed-length prefix approach. Figure 5e shows the number of trie nodes are created for our dataset. When the threshold $d$ on the minimum number of identical prefixes is increased before creating a node, we see that in the beginning there is a steep drop of nodes needed. As in figure 5c, the results are shown for the three different thresholds, the reader now has insight into the balance of the storage size of the trie and the mapping number of keywords per key.

Results concerning keywords alone are not enough to give us insight on what would be the influence on a system with multiple keywords per description. To this end, we have performed numerous experiments to determine the performance of several options on a realistic set of descriptions. To keep complexity to a minimum, we will discuss the results of three comparable parameter settings, one for each option. In figure 3.5 (f), the distribution of query precision can be seen. The number after each setting represents the average number of keywords per key. Our first observation is that it is possible to obtain roughly the same distribution for all three options. Nevertheless, to achieve this, we had to select a parameter setting for the fixed length prefixes with a very low average number of keywords per key. This means that, although we can preserve the privacy of complete queries or descriptions, the privacy of single terms can be defeated. In other words, in the case of fixed length keys, in order to maintain the same precision for queries (i.e. preserve query privacy to an equal degree), we need to sacrifice privacy of individual keywords in the descriptions. This is made evident by the fact that a key in the fixed length approach maps only to 2.2 keywords on average. As far as the secure hash and the information-value based prefix prefixes are concerned, we can see that both have acceptable performance with a reasonable number of keywords per key. Finally, note one more problem of the fixed-length prefix: for approx. 7% of queries, precision is 1, which compromises privacy in a large extent.

### 3.7 Use case: Openknowledge discovery

The OpenKnowledge project aims at knowledge sharing through open web-service interactions. One of its main goals is to build a P2P system, which we call the OK-system [31], for sharing knowledge not only in the form of data, but also in the way data is processed. Using this system, people can publish workflow descriptions (also called Interaction Models or IMs), and peers can subscribe themselves to play

---

4For the complete set of results obtained during our experiments, the reader is referred to an uncompiled version in http://www.few.vu.nl/~kot/privateresources/

5for additional information about OpenKnowledge the reader is referred to: http://www.cisa.informatics.ed.ac.uk/OK/deliverables.html
one or more of the roles in these descriptions. The role-code (i.e. software) that a peer needs to have to play such a role can also be shared and downloaded via the OK-system. Peers can also subscribe themselves to be coordinators of IMs, being responsible for controlling the process flow between the services. All components of the OK-system are implemented in a way that everything runs distributedly without any central control. Sometimes the subscriptions or their constraints need to be kept private, which imposes special requirements on the discovery system. For instance, imagine an interaction model for finding jobs. It goes without saying that it is not acceptable that a subscription to find a new job is visible to all peers in the network, including the current employer.

We will integrate the aforementioned methods in the discovery system of Open-Knowledge to safeguard privacy in subscriptions for participation in interactions. There currently exists a functional implementation of the system\(^6\). More information on the Openknowledge can be found in 9.

### 3.8 Application domain

Resources described by a set of keywords are only an example of the application domain of the model, architecture and methods in this chapter. In this section, we will elicit the requirements for our methods to be applicable and outline other suitable classes of resources.

The basic requirement is that it should be possible to give approximations of descriptors. As a consequence, the range of descriptor components should be large. In our experiments, we have used words from web pages, which have an unbounded range. For resources described by boolean attributes, our methods would be ineffective because boolean values contain too little information to be able to hide part of it. For random identifiers, our methods would perform ideally, since it is impossible to infer an identifier from a part of it.

Apart from Openknowledge discovery, possible applications include the following:

- **Discovery of ontologies** New ontologies are being engineered in an ever increasing pace. Considering their development costs, it may be expected that, in the future, they will be considered so valuable assets that organizations may be unwilling to share. Even nowadays, some institutions are reluctant to share their ontologies. Our methods can be used to publish a flattened ontology as a descriptor. Interested parties would be able to retrieve the location of the system managing the ontology and negotiate further to retrieve the entire ontology (or parts of it) and answer queries. This is possible without having to make the ontology public.

\(^6\)www.openk.org
• **Image search** Practically all image descriptors (e.g. histograms) can be approximated, for example by rounding or truncating values. It is also possible to approximate image queries in a similar fashion.

• **Resources described by attribute-value pairs** Resources can be described using attribute-value pairs. The methods presented in this chapter are directly applicable in this domain.

### 3.9 Conclusions

In a world where data privacy is becoming increasingly important, and a field (peer-to-peer systems) where little has been done to protect it, we have taken a first step towards developing a scalable privacy-preserving discovery system by outlining the fallacies of current systems, proposing a framework and an architecture based on DHTs and evaluating settings for our proposed methods. Our framework follows a peer-to-peer paradigm, where peers publish descriptions of their resources to an index and are contacted directly by interested peers for additional information. We have proposed the use of a DHT maintained by a network of untrusted peers to maintain this index and three methods to hide the contents of the descriptions from the peers in the DHT. The first method is to use a secure hash, like SHA-1, to generate keys for the keywords in each description. The results show that this method gets very close to the desired ‘horizontal line’ when looking at the distribution of mapping keywords per key. The disadvantage of this approach is that the hash algorithm does not preserve keyword similarity (e.g. ‘computer’ and ‘computers’ hash to completely different keys), which makes approximate matching impossible. The second method we investigated is to use a fixed ratio of the prefix of each keyword as a key. Although we now have a similarity preserving representation, it comes with the price of a high variance on mapping keywords per key because some prefixes are much more popular than others. The disadvantage of a high variance is that it could result in some cases only very few or only one keyword maps to a key, revealing the intentions of the querier or storing peer. A high variance can also lead to a high network load because when a key maps to many keywords, all peers registered for the key will be returned to the querying peer. The last proposed method calculates the information-value (i.e. popularity) of each keywords and uses it in determining the prefix length. The assumption is that we have access to a representative set of keywords to generate a trie which is an space efficient data-structure to store the popularity of prefixes. The results of the information-based approach indicate that the last method resembles much more the ideal ‘horizontal line’ than the method using a fixed prefix ratio.

We have provided insight in the right parameters for these methods and their applicability. Future work lies in analyzing the influence of different types of data (e.g. image histograms or concepts from ontologies) on the performance of the methods presented in this chapter.
Acknowledgements

We would like to thank Frank van Harmelen for making very useful comments and suggestions and Anna Tordai for proofreading. This work was supported by the EU OpenKnowledge Project (IST-2005-027253).
Chapter 4

Scalable rendez-vous

Many data processing problems require that "Items with some given key are collocated at some point in time". When key distributions are very skewed, to the point that one host cannot possibly hold all the items for a given key, current approaches (e.g. DHTs) do not scale.

With regard to the goals set in the introduction, this chapter focuses on scalability and completeness (since the algorithm will eventually collocate all items with the same key).

This chapter is based on "Marvin: distributed reasoning over large-scale Semantic Web data" to appear in the Journal of Web Semantics.
CHAPTER 4. SCALABLE RENDEZ-VOUS

abstract

We address the problem of making scalable and load-balanced rendezvous points. We present SpeedDate, which combines data clustering with random exchanges. The random exchanges ensure load balancing, while the data clustering attempts to maximise efficiency. SpeedDate is compared against random and deterministic (DHT-like) approaches, on performance and load-balancing. We simulate parameters such as system size, data distribution, churn rate, and network topology. The results indicate that SpeedDate is near-optimally balanced, performs in the same order of magnitude as a DHT-like approach, and has an average throughput per node that scales with $\sqrt{i}$ for $i$ items in the system.

4.1 Introduction

In this paper we present a solution for scalable rendezvous peers, using randomised exchanges and data clustering. Rendezvous peers [118] are used in many systems, such as content-based publish/subscribe systems [90] and event dissemination systems [25]. We target a common problem in event-based models, namely making rendezvous points scalable and load-balanced; this problem is also well-known from a game-theoretical perspective [5, 67].

The problem that we address is as follows: we have a system of several nodes; each containing some number of data items. Each item has a key, and multiple items may share the same key. The number of items per key is very unevenly distributed, and typically follows a power law. Items want to pairwise meet all other items with the same key (their “buddies”); two items meet if they are both located at the same node at some point in time. Our system is consistent if and only if all items have met all other items with the same key.

Deterministic rendezvous points for each key (for example, using a distributed hashtable) suffer from load-balancing problems. Our solution is called SpeedDate, which can be understood as a rendezvous mechanism in which data is clustered around fuzzy rendezvous points. SpeedDate is:

- scalable, as opposed to random exchanges,
- load-balanced, as opposed to deterministic (DHT-like) approaches, and
- decentralised, as opposed to broker-based approaches.
4.2. MOTIVATION

In this thesis, we will only deal with load balancing in terms of storage, i.e. balancing the amount of data that needs to be stored by each node. Depending on the application, this may be translated to balancing execution time across nodes.

In the following, we first present the motivation for this work and explore related work in this area. We then present our “speed-dating” approach and evaluate its performance under various parameters such as data distribution, network scale, overlay type and churn.

4.2 Motivation

The Web of Data [109] is a collection of large semi-structured datasets: large graphs of interlinked statements. Many billions of interlinked statements can now be found online, covering domains such as geography (Amsterdam locatedIn Netherlands), politics (Obama presidentOf USA), or life sciences (geneXXX encodesFor lungCancer). All these statements are represented in a common data-model called RDF, where each statement is called a “triple” [20]. Through logical reasoning these graphs can be expanded with implicit information. For example, by combining (Amsterdam locatedIn Netherlands) and (Netherlands locatedIn Europe), we can derive (Amsterdam locatedIn Europe). Given the size of the data, scalable reasoning is a major challenge [106, 44]

We adopt a distributed reasoning approach [83]. For distributed reasoning, triples that share common elements (“Amsterdam” in the example above) should be co-located at the same machine and combined into additional triples. The challenge in this scenario lies in assigning rendezvous points for triples. Existing approaches extract three keys for each triple (namely the three elements of the triple), and use those keys to assign rendezvous points. However, since the data distribution of these elements is highly skewed, such approaches will suffer from load-balancing problems on the physical hosts where the rendezvous points are located.

4.3 Related work

We give an overview of related work, starting from random rendezvous points (which are balanced but inefficient), to deterministic rendezvous points (which are efficient but unbalanced) and rendezvous regions (which do something in the middle).

4.3.1 Random rendezvous points

To find an arbitrary rendezvous point for two data items, a straightforward approach is to partition and exchange all data randomly [67, 83]. That way, items with the same key will eventually meet, namely if they happen to be located at the same node. This approach ensures load balancing, but is inefficient and does not scale
well: the encounter probability of triples decreases as the number of keys and nodes increase.

Random rendezvous points have been studied from a theoretical perspective \cite{5} in the specific situation where items have no key and want to meet all other items. In our case, items have different keys, and they want to meet others with the same key. Random rendezvous points ignore all keys, while we will use the keys to improve efficiency (but without disturbing load balance).

### 4.3.2 Deterministic rendezvous points

Deterministic rendezvous points are used in many distributed systems, including structured peer-to-peer overlays such as distributed hashtables (DHTs) \cite{72}. Rendezvous points are used for key-based data storage and retrieval, where keys are used to determine the rendezvous points.

Deterministic rendezvous points are also used in topic-based publish-subscribe systems such as Scribe \cite{25} or Minstrel \cite{53}: subscriptions and publications meet at rendezvous points which are based on their topics. Subscribers are notified of each successful encounter, for example through a dedicated group overlay \cite{25}

Several distributed reasoning techniques have been proposed based on deterministic rendezvous points \cite{21,59} using an underlying DHT. Here, each triple is sent to three rendezvous peers (one for each of its terms: subject, predicate, and object), which ensures that triples with common terms will be co-located. However, given the size and distribution of the data (many billions of triples, with terms occurring according to a power-law \cite{82}) the rendezvous peers will suffer from highly unbalanced load distributions.

Note that standard techniques for load-balancing \cite{60} will not work in our situation, since: (a) we have more items sharing one key than can fit in a single node, so replication and caching will not help, and (b) we need all items with the same key to meet each other, so sub-dividing the keyspace over multiple responsible nodes will also not help.

### 4.3.3 Geographic rendezvous regions

In \cite{107}, “rendezvous regions” are introduced, an approach for distributed data storage in mobile sensor networks. They partition the key space into geographical regions. The nodes that are physically located inside one region elect several “server” nodes that are collectively responsible for storing and retrieving data in that region, and replicate all data between them. The approach is similar to a hierarchical superpeer overlay: the “server” nodes are replicated superpeers \cite{133}, all regular nodes forward their queries to these superpeers. In contrast to our work, their region detection is deterministic, and rendezvous inside each region are performed through flooding.
4.4 Our approach: SpeedDate

Let us recall our situation: we have a set of items with particular keys, multiple items may have the same key. We also have a set of nodes, each of them being able to store a number of items. The nodes do not have any special rights over items: any node can store any item.

Our goal is to have as many items encounter or meet their “buddies” (other items with the same key). By encounter we mean: the items must at some point in time be located in the same node. We do not care whether keys are always at the same node, we just want, over time, that items meet others with the same key. Furthermore, encounters have at least once semantics, we are not interested in how many times an items encounters another item with the same key, as long as they meet at least once.

In other words, the items want to speed-date each other. There are many rooms (nodes) that can hold only a finite number of items at a time. Items want to meet as many other items with the same interest (key) as possible yet minimise the time spent travelling.

The consistency criterion for our system is that all items with the same key have encountered each other. In our case, this means being collocated on the same node at some point: \( \forall k \in \text{keys}, \forall i, j \in \text{data}_k : \exists t, n : \text{located}(i, n, t) \land \text{located}(j, n, t) \).

In this paper, we focus on finding rendezvous points; what happens after rendezvous is orthogonal. Publish/subscribe or event dissemination systems could for example be built on top of our approach, by using some other mechanism (e.g. a group overlay) to notify subscribers of successful encounters.

4.4.1 Intuition

Our approach combines the balanced load of random exchanges and the efficiency of deterministic rendezvous nodes. Instead of a deterministic rendezvous location for each key, data is moved around randomly, but with a (strong) bias towards some nodes. This bias is determined using item keys and node identifiers (similar to distributed hash tables). The bias improves clustering and encounter probability, while the random exchanges ensure that items are distributed evenly among the nodes. Let us illustrate the algorithm with an example.

<table>
<thead>
<tr>
<th>node</th>
<th>initial items</th>
<th>eventual items (at some time ( t ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( p, s, s )</td>
<td>( p, q, r )</td>
</tr>
<tr>
<td>( B )</td>
<td>( r, t, u, v )</td>
<td>( s, s, t, u )</td>
</tr>
<tr>
<td>( C )</td>
<td>( q, s, t, t )</td>
<td>( s, t, t, v )</td>
</tr>
</tbody>
</table>

Table 4.1: Data distribution over nodes
Example 1 We have three nodes, A, B and C, and a set of items, with an uneven distribution of keys. Initially, items are distributed randomly amongst nodes, as shown in Table 4.1. We can position node identifiers and item keys on a space that wraps-around (e.g. using a hash function and modulo arithmetic), as shown in Figure 4.1. In this space, the position of A, B, and C signifies their node IDs. Items are positioned according to their key, not according to their current location: their location is indicated graphically with subscripts, e.g. $p_a$ means item $p$ is located at node A.

In our algorithm, each node autonomously selects some other node (the selection mechanism will be explained later), and exchanges data. For example, A will ask B for one item. When asked by A, B will return the item it has whose key is closest to A on the common key and node identifier space. In this case item, B will return $r_b$ (the items $p$ and $q$, which are closer to A, are located at nodes A and C). In return, B will receive an item from A that is closest to B, in this case one $s_a$.

After exchanging several data items, the data distribution will start clustering, as we can see in the lower part of the figure. Since nodes keep asking each other for data, the distribution never converges but keeps oscillating around a clustered state. Eventually (shown in the right column of Table 4.1), node A will mostly contain the items $p$, $q$, and $r$, B will mostly contain $s$, and C will contain $t$, $u$, and $v$.

We can see that through biased random exchanges, a responsibility area emerged for each node, and was adjusted to ensure a balanced distribution of the initial data. These responsibility areas are an emergent property of the algorithm and are not explicitly defined.

In contrast, a DHT-like approach would use rigid responsibility areas (based on item key and node ID) and thus assign most items to node B, leading to load-balancing problems (see Section 4.3.2).

We will now describe our SpeedDate rendezvous algorithm, which was used in the example above. Since our algorithm is a mixture between the efficient ap-
4.4. OUR APPROACH: SPEEDDATE

Algorithm 4 Random rendezvous

I: set of my items
P: set of IDs of my neighbouring nodes

procedure MAIN
  while true do
    i ← \text{pick\_uniform\_random}(I)
    p ← \text{pick\_uniform\_random}(P)
    send(i, p)
  end while
end procedure

proach of deterministic rendezvous points and the balanced approach of randomised rendezvous points, we first briefly explain these.

4.4.2 Reference approaches

A deterministic rendezvous algorithm uses some deterministic mechanism to map item keys to rendezvous points maintained by rendezvous nodes. The convergence criterion would be met quickly and efficiently, since all items with the same key would be sent to a given rendezvous, where they would meet each other.

It is straightforward to implement such a function with a DHT or with a broker. The main drawback of this approach is that it suffers from load balancing issues: What if there are more items for a given key than any single host can hold?

Nodes may also exchange items randomly, shown in Algorithm 4. At every time step, they select a random subset of their items and send it to a randomly selected node. Both nodes and items are picked according to a uniform random probability function, i.e. all nodes and items have equal probability of being picked. Here, we assume that the network is fully connected, i.e. nodes can send messages to any other node. Section 4.5 will also evaluate the performance of random exchanges in partially connected networks.

Since the set of buddies encountered after each exchange is random, the method meets our eventual consistency criterion. The random approach does not suffer from load-balancing problems, since all nodes have an equal probability of receiving an item.

4.4.3 SpeedDate

The SPEEDDATE algorithm is a hybrid between a random exchange and a deterministic rendezvous. Nodes are assigned a random ID in the same space with the keys using a uniform distribution. Unlike DHTs, this ID does not need to be unique. Furthermore, every node has a set of neighbours, of which it knows the ID. Again,
we can first assume that the network is fully connected; Section 4.5 will evaluate the performance in partially connected networks.

Similar to the random approach, nodes ask their neighbours for items. The uniqueness of SpeedDate lies in:

- Instead of returning random items, nodes return the items that are closest to the ID of the asker. This enables data clustering, thus increasing performance.

- Instead of picking random neighbours for exchange, nodes prefer neighbours with ID close to their own. This ensures that data remains clustered, and improves probability of data encounters.

- Instead of only returning optimal data items, nodes always return some fixed amount of items when asked. If nodes do not have any items with key close to the key of the asker, they will still return the best they have. This ensures that the system is load balanced.

The SpeedDate approach is formally described in Algorithm 5. Each node repeats the following eternally:

1. Pick a neighbour using a Gaussian (normal) distribution on the node ID. Each node maintains a Gaussian distribution, with mean $\mu$ on his own ID, and some
given variance $\sigma$. When run with low variances, nodes will mostly select their close neighbours for exchange (whose ID is close to their own). With high variance, nodes will exchange nodes more uniformly across all other nodes.

2. Pick the items with keys closest to the ID of the selected neighbour. Each node contains a set of items, and each item has one key. Having selected one neighbour for exchange, nodes will select the data item whose key is closest to the ID of that selected neighbour.

Note that this selection does not necessarily entail iterating over all items: nodes could pre-process their items into buckets or use sorted data structures such as trees.

3. Send those items and receive some of the neighbour’s items in return. Nodes exchange items symmetrically, which ensures load balancing (all nodes have the same amount of data). The neighbour selects items according to the same principle as explained above, thus returning items whose keys are closest to the sender’s ID.

4.4.4 Visual comparison of data clustering

The three algorithms are visually compared in Figures 4.3–4.4. The images shown are snapshots of the data distribution for a small example of 100 nodes, 100 keys and 10000 items. They are taken at after 33, 66 and 100 time units respectively.

The horizontal dimension shows the nodes, the vertical dimension shows the keys. Brightness represents the number of items with the given key at the given node. Key popularity is unevenly (linearly) distributed: not all keys appear in the same number of items. In the images, keys are sorted according to increasing popularity: keys at the bottom of the image are more popular than keys at the top.

In the random approach, shown in Figure 4.2, items with the same key are dispersed throughout the nodes, and the concentration of items in one node is never large enough to produce a bright pixel (the picture looks almost completely dark, but is not). Although this uniform distribution is not good for performance, all nodes have a similar load.

In the DHT-based approach, shown in Figure 4.3, items quickly move to the nodes responsible for them (optimal distribution already reached at the second snapshot). This results in a strongly partitioned data distribution, all items with the same key are located on the same node, producing bright white pixels. However, as we will show in Section 4.5, for realistic data distribution the partitioning results in poor load balancing (not obvious in the figure).

Figure 4.4 shows snapshots of the data distribution in SpeedDate. Over time, all items with some key gather in the neighbourhood of one node, increasing their encounter rate. Nodes share responsibility for a key, resulting in better load balancing. Furthermore, notice that the horizontal lines for the more popular key (i.e. the
keys at the bottom of the image) are longer than those in the top, meaning that more nodes share responsibility for keys with more items.

Figure 4.2: Simulated random data distribution (at 33, 66, and 100 time units respectively)

Figure 4.3: Simulated data distribution in DHT (at 33, 66, and 100 time units respectively)

4.5 Evaluation setting

In this and the next section, we evaluate the performance of SpeedDate compared to the presented two reference approaches (random and deterministic exchanges). We also explore the parameter space of our algorithm to determine sensitivity and optimal settings.

We have used a purpose-built simulator for our experiments. The simulation proved to be computationally demanding, so we have used a quad-processor 2.3GHz server-class machine with 32GB of main memory. We use a simulation clock. Every clock cycle, nodes may send a fixed number of messages.
4.5. EVALUATION SETTING

Figure 4.4: Simulated data distribution in SPEEDDATE (at 33, 66, and 100 time units respectively)

4.5.1 External parameters

Experiments were performed using an artificial dataset, under the following conditions:

Nodes We have $n$ nodes in the system; all nodes are considered to be homogeneous, with same functionality and specifications. Nodes have limited capacity for items which cannot be exceeded. Nodes have a limited bandwidth of messages per clock cycle.

Items There is a total of $i$ items in the system. Each item has one (non-unique) key. There is a total of $k$ unique keys in the system.

Key distribution The number of items with a given key follow one of the following distributions:

- **Uniform** All keys appear in the same number of items: $\text{items}(key) = c$, with $c = \frac{\#\text{items}}{\#\text{keys}}$.

- **Linear** The number of items per key are linearly distributed: $\text{items}(key_r) = c \times r$ where $r$ is the popularity rank of the key, for some constant $c$.

- **Zipf** The Zipf distribution for $n$ outcomes is defined as $\text{Zipf}_n(r) = \frac{1}{r \times \sum_{j=1}^{n} (1/j)}$, where $r$ is the rank of the outcome (ordered by probability), and is a realistic distribution for Web documents and Semantic Web data[82]. Note that the probability density function is very steep: for 5000 outcomes, the top-3 has a total probability of more than 20%.

Churn We assume a fail-silent model for nodes. I.e. in every cycle, each node has a fixed probability $\text{churn}$ of being offline. This means that no messages are sent or received by that node. Message senders do not know whether nodes are online, so there is a chance that messages are lost. In this case, we assume
that the sender can detect that the receiver did not receive the message and retain the items that would be sent in its own data. In our setting, churn is the inverse of availability.

4.5.2 Internal parameters

Our algorithm has the following parameters:

Connectivity Each node keeps a connection to a number of other nodes, ranging from 1...n, the latter being a fully connected network. The higher this number, the more maintenance is required on the overlay network.

Topology When the network is not fully connected, neighbours are selected either randomly or according to their ID. We will refer to the former topology as “random” and to the latter as “proximity”. In the proximity topology, nodes prefer neighbours with ID similar to their own, resembling the topology of the Chord DHT[119].

Neighbour bias The variance $\sigma$ represents the bias of nodes to select neighbours with similar IDs to their own for exchanges. For readability, we show the inverse: $1/\sigma$. A high $1/\sigma$ means highly selective bias: nodes mostly select peers close to them.

4.5.3 Evaluation criteria

We use the following criteria to evaluate our algorithm:

Recall per key We measure recall of encounters as ratio of the actual encounters vs. possible encounters for each key:

\[
\text{recall} = \frac{1}{K} \sum_k \left( \frac{\text{actual encounters for key } k}{\text{possible encounters for key } k} \right)
\]

Recall per item We also calculate recall per item to indicate system behaviour towards more-popular keys (which would normally overpower the normal, per key, recall):

\[
\text{w. recall} = \frac{\sum_k (\text{actual encounters for key } k)}{\sum_k (\text{possible encounters for key } k)}
\]

Load balance We measure, at each clock cycle, the data load in each node and the number of messages sent and received. We evaluate in terms of maximum and standard deviation of $\text{load}_d$ and $\text{load}_m$ across all nodes.

Unless mentioned otherwise, all experiments use the default parameters from Table 4.2 which were obtained through a series of exploratory experiments.
4.6. RESULTS

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr. items</td>
<td>100,000</td>
</tr>
<tr>
<td>nr. keys</td>
<td>5,000</td>
</tr>
<tr>
<td>nr. nodes</td>
<td>500</td>
</tr>
<tr>
<td>nr. neighbours</td>
<td>( \log_2(\text{nr. nodes}) )</td>
</tr>
<tr>
<td>key distribution</td>
<td>Zipf</td>
</tr>
<tr>
<td>topology</td>
<td>proximity</td>
</tr>
<tr>
<td>selection bias ( \sigma )</td>
<td>0.25</td>
</tr>
<tr>
<td>node capacity</td>
<td>200</td>
</tr>
<tr>
<td>node bandwidth</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Default parameters

<table>
<thead>
<tr>
<th>approach</th>
<th>( \sigma_{data} )</th>
<th>max(data)</th>
<th>( \sigma_{msgs} )</th>
<th>max(msgs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>13.18</td>
<td>242</td>
<td>10.48</td>
<td>244</td>
</tr>
<tr>
<td>deterministic</td>
<td>620.99</td>
<td>11021</td>
<td>316.71</td>
<td>5685</td>
</tr>
<tr>
<td>SpeedDate</td>
<td>14.08</td>
<td>242</td>
<td>12.52</td>
<td>281</td>
</tr>
</tbody>
</table>

Table 4.3: Load-balancing (lower is better)

4.6 Results

Before experimenting with our algorithm under various settings, we compare SpeedDate with two reference approaches: one where random nodes exchange items at random (with very good load balancing properties) and a deterministic approach (with efficient rendezvous) where DHT nodes responsible for a key are used as rendezvous points.

Figure 4.5 shows our approach, compared to the random and deterministic rendezvous approaches. The figure shows, on the left, per-key rendezvous recall over time, and on the right, per-item recall over time. We can see that in both cases, the deterministic approach outperforms SpeedDate and that SpeedDate clearly outperforms the random approach. Note that the per-key recall of SpeedDate is higher than the per-item recall, which means that we favour rare items over very popular items.

Table 4.3 compares the load-balancing properties of these approaches, showing the standard deviation and maximum number of items stored and messages received per node. The presented values are the highest across all cycles in one simulation.

We can see that the random approach is clearly load-balanced, both in terms of data load and messages received (low deviation, low maxima). The deterministic approach suffers from severe imbalance, one node stores over 11k items while the average is 200. In reality, this would pose serious scalability issues. The same holds for message load. Load balance in SpeedDate is comparable to random, with low
deviation and low maxima.

In short, we can see that SpeedDate combines the efficiency of a deterministic approach with the load balance of a random approach.

![Graphs showing comparison of approaches, using (a) default recall and (b) per-item recall.](image)

**Figure 4.5:** Comparison of approaches, using (a) default recall and (b) per-item recall

### 4.6.1 Topologies

We evaluate our algorithm under various overlay topologies and number of neighbours per node. Furthermore, we want to find the optimal values for neighbour bias \( \sigma \). Table 4.4 shows these results which are also summarised in Figure 4.6a. From these we can conclude the following:

**Topologies** Proximity and full clearly outperform the random topology. It is not clear whether proximity outperforms full.

**Number of neighbours** The proximity topology is not very sensitive to the number of neighbours per node. We can see that it has similar performance for 3 neighbours per node up to 500 neighbours per node (fully connected network). The fact that only 3 neighbours are required to achieve acceptable performance is very encouraging given the maintenance cost for neighbours.

**Sensitivity to the value of sigma** The proximity topology is not very sensitive to the setting of neighbour bias \( \sigma \). We can see that the system produces similar results for a \( 1/\sigma \) ranging from 0.1 to 5. Note that the fully connected network performs best when we use a very steep distribution to select neighbours. The proximity topology performs well even if we use an almost uniformly random distribution. This is attributed to the fact that the set of neighbours already contains the nodes with the closest ID to the node.
### 4.6. RESULTS

The horizontal axis shows $1/\sigma$ to enhance readability. Simulations were limited to a maximum of 500 cycles. Lower is better.

Note that for very low sigmas (high $1/\sigma$), the proximity topology performs very badly, which is attributed to the symmetry of the topology (because nodes select their left and right neighbours using a symmetric function). For a very high selection bias, nodes will only exchange items pair-wise with one neighbour, which has a negative effect on recall.

#### 4.6.2 Data distribution

Figure 4.6b shows the recall rate over time for different data distributions. Initially, the algorithm performs slightly better on data following a Zipfian distribution, which we attribute to it favouring rare keys (which appear more frequently in the Zipfian distribution). Overall, the algorithm performs slightly better on uniform and linear distributions.
distributions, which we attribute to the large amounts of potential encounters that must be found for popular data.

### 4.6.3 Churn

Figure 4.7 shows how our system performs under node failures. We use the default settings except for churn, which ranges from 0% to 90%. Figure 4.7a shows recall over time, Figure 4.7b shows time needed to reach 90% recall.

For every node that is unavailable we lose the messages that it would send plus the messages that it would receive. This amounts to a loss in messages equal to the square of the loss in availability: for example, for 50% availability, only half of the nodes will send messages, out of which half will be lost (since the receivers are down), amounting to 25% of the messages that would have been sent with 100% availability.

In general, the ratio of the messages delivered in a system with availability $x$ to a system with 100% availability is $x^2$ ($x$ are sent, of which $x$ arrive); since churn $\alpha$ is defined as $1 - x$, messages delivered as function of churn is $1 - (2\alpha - \alpha^2)$. In a perfect system without recovery mechanism, loss in messages would exactly equal loss in recall (all undelivered messages are lost encounter opportunities).

Indeed, our simulation results indicate that the performance loss in SpeedDate, in the presence of peer failures, follows this model. Figure 4.7b compares our simulated results against this prediction.

The construction and maintenance of the overlay network is orthogonal to our approach; thus coping with permanently failed connections is outside the scope of this paper.
4.6. RESULTS

Figure 4.7: Recall with increasing churn

4.6.4 Scalability

In evaluating the scalability of our algorithm, we have made the following assumptions: (a) node capacity stays constant, thus as the number of items increases, so does the number of nodes; and (b) as the number of items increases, so does the number of keys. Note that Zipf is a scale-free distribution; thus, with an increasing number of items, item popularity still follows the same distribution.

Figure 4.8 summarises our results, showing recall over time on the left and time needed to reach 90% recall on the right.

Note that the different settings in Figure 4.8a do not show a linearly increasing number of items. In fact, we notice that for $20 \times$ the number of items, we have $5 \times$ worse performance, for $3000 \times$ more items, we have $10 \times$ performance loss, indicating good scalability properties.

As indicated in Figure 4.8b, the performance of our algorithm as a function of growing system size, seems to follow a curve of $\sqrt{i}$, where $i$ is the total number of items in the system.

4.6.5 Anytime behaviour

As we can see in the Figures 4.5–4.8, recall does not increase linearly with time. Instead, we can observe three phases in the algorithm: first the clustering phase in which data is being moved towards their target neighbourhood, then the exploitation phase in which items are exchanged within clusters and encounter rate is high, and a final phase where the remaining items slowly fight all odds.

SpeedDate has good anytime behaviour: in all settings, 80% of the results are produced in the initial and exploitation phase, which take around one-third of total running time.
CHAPTER 4. SCALABLE RENDEZ-VOUS

4.7 Conclusion

We have developed SPEEDDATE, a scalable and load-balanced rendezvous mechanism. The benefits of our method are: (a) it is load-balanced, as opposed to DHT-based approach, having 40\times smaller standard deviation in data load per node, (b) it clearly outperforms a random approach by orders of magnitude, performs within 3\times of the DHT-based approach and scales in general with \sqrt{i}, (c) it functions even using a small number of connections, namely 3, per node, (d) it is robust against failures and handles churn rates of up to 50% with less than 3\times performance loss, (e) it is simple to implement and does not require expensive network maintenance.

The motivation for this work comes from the field of distributed reasoning, but we believe that our approach is applicable to more domains: the robustness against node failures, the limited number of neighbours required and the scalability make SPEEDDATE a good candidate for e.g. sensor networks.

Figure 4.8: Recall with increasing system size

(a) recall over time
(b) time to reach 90% recall
Part II

Infrastructure
Largely based on the algorithms described in part 1, four infrastructures are described in this part.

Chapter 5 refers to scalable web-service discovery and chapter 6 to privacy-preserving sharing, with a primary focus on control.

Chapters 7 and 8 refer to reasoning with Semantic Web data. They focus on two approaches for distributed reasoning: Chapter 7 follows a peer-to-peer paradigm where peers keep their data local. Combining data is done on a by need basis, meaning that once a query is posted, the peers will communicate to combine their data (i.e. data combination is triggered by queries).

In the approach described in chapter 8 data is equally split across all nodes. Combining data is done a priori (i.e. data combination is triggered by new data entering the system).
Chapter 5

Web service discovery

In collaboration with SeekDa inc., we have applied the rarity-based technique presented in chapter 2 to Web Service discovery and evaluated it against a real-world web service description corpus.

With regard to the goals set in the introduction, this work pertains to scalability, interoperability and openness.

This chapter is based on the paper *Massively scalable web service discovery, presented in the IEEE 23rd International Conference on Advanced Information Networking and Applications (IEEE AINA-09)* [6].
The increasing popularity of Web Services (WS) has exemplified the need for scalable and robust discovery mechanisms. Although decentralized solutions for discovering WS promise to fulfill these needs, most make limiting assumptions concerning the number of nodes and the topology of the network and rely on having information on the data a-priori (e.g. categorizations or popularity distributions). In addition, most systems are tested via simulations using artificial datasets. In this chapter we present a lightweight, scalable and robust WSDL discovery mechanism based on real-time calculation of term popularity. Results based on a large-scale emulation on the DAS-3 distributed supercomputer, using real data from seekda, show that we can achieve web-scale service discovery based on term search.

5.1 Introduction

In the past, Web Services (WS) were mainly used in the context of corporate environments and online commerce. Nowadays, we also see a plethora of other devices supporting Internet connectivity. Already in 2003, Forrester Research founder and chief executive officer George Colony, claimed that “about 500 million devices are connected to the Internet, but by the end of the decade there would be billions of connected devices, including cars, phones and many other electronic devices.” \(^1\) Most of these devices will be able to produce information accessible through WS. Developing an infrastructure for such large numbers of services is a pragmatic need and as such, a promising research domain.

The most widespread solution for storing and indexing WS descriptions is the UDDI standard [128]. UDDI was originally perceived as a centralized repository and thus suffers from the associated drawbacks, namely being a single point of failure (SPoF) and being unable to scale gracefully.

The latest UDDI specification (v3.0.2 released in 2004) includes the option for a cluster of UDDI registries and specifies interactions between them. UDDI nodes function as mirror replicas - an approach that addresses the single point of failure but is not efficient, as all nodes need to be kept synchronized.

In academia, other distributed solutions have been proposed to overcome issues of scalability and robustness, and to push down cost. Most of these approaches rely

\(^1\)Computer Weekly, March 11, 2003 (http://www.computerweekly.com/Articles/2003/03/11/193049/forrester-ceo-forecasts-web-services-storm.htm)
on a-priori semantic agreement, which is usually expressed as a shared ontology or a categorisation [131, 104, 87, 125] used to create the network topology.

Evaluation of these systems has not been thorough, as new methods were tested with artificial and unrealistic data. This is attributed to the fact that until recently, few publicly available WS description corpora were available. Moreover, rarely have we seen an evaluation going beyond simulation experiments on the algorithms.

In this chapter we present a fully functional discovery system that improves scalability by using term popularity to optimize indexing while also calculating these statistics in real-time, meaning it requires no globally shared knowledge and self-adapts to the dataset. Furthermore, we evaluate our method using a real-world dataset, obtained by processing the information collected by the seekda WS search engine\(^2\), without the need to attach additional annotation to WSDL files. This dataset consists of a set of terms describing WS and a sample of anonymised queries posted on seekda. Evaluation was performed using hundreds of real nodes under heavy load in the presence of node failures.

5.2. Going distributed: Peer-to-Peer systems, DHTs and Multi-term search

Since we are aiming at dealing with the issues of scalability and SPoF vulnerability for WS registries, we choose to pursue the distribution approach: namely, by building a completely decentralized discovery service we are able to cope with increasing load while also removing the SPoF vulnerability of centralized approaches. We explain the building blocks of our approach here and discuss related approaches in Section 5.5.

5.2.1 Peer-to-Peer

Peer-to-Peer (P2P) overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. Peers form self-organising networks that are overlayed on the Internet Protocol networks and go beyond services offered by client-server systems by having symmetry in roles, whereby a client may also be a server. By doing so they are able to display features such as a robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance.

5.2.2 Single-term search via DHTs

Distributed Hashtables (DHTs) are a particular subclass of P2P systems aimed at storing content in a completely decentralized way [73]. Nodes function autonomously

\(^2\)http://www.seekda.com/
and collectively form a complete and efficient system without any central coordination. In DHT overlays, each object is associated with a key, chosen from a large space. This space is partitioned in bins, and each peer is responsible for the keys and corresponding objects in the respective bin. Peers need to maintain connections only to a limited number of other peers and the overlay has the ability to self-organize, with respect to peer connections and object distribution, to handle network churn.

In principle, all DHT-based systems provide the following functionality: \texttt{store(key, object)}, that stores an object identified by its key, and \texttt{search(key)}, that returns the object (if found) from the peer responsible for the key. Peers communicate asynchronously via messages; current systems need approximately $O(\log(N))$ messages to search or store and each peer needs to keep from $O(1)$ to $O(\log(N))$ pointers to other peers, where $N$ is the number of peers in the network.

Although DHTs are robust and have very efficient key lookups, their application domain is limited by the absence of efficient methods to search for richly described content. They do however provide a substrate on which more sophisticated search capabilities can be based.

\section*{5.2.3 Multi-term search}

So far we have described how a DHT can be used in order to enable efficient single term search. However, for settings in which resources are described and discovered using multiple terms, such as WS discovery based on WSDL description annotations, we need to support multiple term search.

We assume a setting in which we want to support multi-term conjunctive queries\textsuperscript{3}. A naive approach for multiple term search would be to maintain a distributed inverted index over all terms in a DHT:

- **Insert** new resources into the system by hashing each term and storing it together with a pointer to the resource in the DHT overlay.
- **Retrieve** resources by performing a lookup on the hash of each term of a query and then performing a local join on retrieved results.

Unfortunately, this approach would not perform well, for a number of reasons:

- **Distributed join.** To perform a distributed join, the initiating peer has to gather all index entries for all the terms in the query. The cost of this can be prohibitively high, especially for queries with many terms. The problem is further aggravated if some terms are much more popular than others. In document retrieval systems, terms usually follow a zipf distribution \cite{19}. Therefore, a query with at least one of these popular terms will be very expensive to calculate, since it would imply retrieving every description mapped to those terms.

\textsuperscript{3}Disjunctive queries are easier to support as they can be split into a set of single-term queries
5.3. OUR APPROACH

- **Load distribution.** The fact that term frequency distribution usually follows a Zipf pattern can cause severe load distribution problems, since the bins (i.e., the peers in the DHT) mapping to very common terms would have to store a large number of descriptions and process a large fraction of the total queries. One way of dealing with this would be to set a threshold on the number of descriptions that a peer can store for each term. When this threshold is reached, new descriptions for that term will not be stored anymore. Although this would solve the storage problem of peers responsible for popular terms, the message processing load caused by requests to index and query descriptions will still be very uneven. To make matters worse, blindly discarding terms reduces recall, especially for rare terms, since popular terms are more likely to be used in indexing anyhow.

- **Large descriptions.** So far we have assumed that descriptions are replicated to all the peers responsible for each of their terms, which poses another potential issue: what will happen if these descriptions are substantially large? As we shall see in Section 5.4, it is not uncommon to find service descriptions characterized by hundreds of terms. In this approach, the number of messages and replicas increases linearly with the number of terms in the description, which makes the approach non-scalable when objects have a large number of terms.

5.3 Our approach

We have developed a popularity-based approach for indexing and discovering WS descriptions consisting of possibly large sets of terms.

We already described the scalability problems when using a traditional DHT approach by indexing all terms. Since WS can be described by a large number of terms, we need a method that scales gracefully with the number of terms per description.

Our approach is based on the work on rarity-based routing presented in chapter 2. The key idea is that some terms occur more frequently in descriptions than others, hence the more popular a term the bigger the chance it will be found by a random walk through the peers in the DHT. By having peers in the DHT keep statistics about the popularity of the terms, we can use this information to put more indexing effort in the rare terms.

In [116], the authors suggest that for queries on common items flooding queries is sufficient, while for rare items DHTs perform best. Indeed, for common terms we are not interested in getting all results, if there are millions of them; the first $N$ would be enough. On the other hand, for rare terms, we are interested in all results. Our approach is to use statistical information to determine rare terms and queries that refer to them, thus adapting the routing process accordingly.

This approach has also been taken in recent research efforts [71, 55], albeit with no efficient way to determine which items are rare. Typically, popularity-based
approaches assume prior knowledge of which items are popular, which is unrealistic. But how do we determine in an efficient and scalable way whether a term is rare or not? Unlike previous approaches, we do not assume external or centralized sources of statistical information, but rely on the statistical information calculated automatically, distributively and dynamically by the individual peers in the DHT.

WSDL files in our system are described by a set of terms, which we call descriptors. These descriptors will be placed in bins, i.e. peers where one of the terms hashes to. Each of these bins will contain many (or all) descriptors for this specific term. Thus, every description will be potentially stored by more than one peer and every peer will potentially store more than one description. We will show that this information alone is enough to calculate statistical information (e.g. we can calculate the popularity of a term) that will help us index descriptions more efficiently\(^4\).

In other words, the uniqueness of this approach is that it exploits the structure of DHTs to extract statistical information useful for routing.

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

To illustrate our case, imagine the following: In the simple approach described in Section 5.2, the description for an example Stock Quote\(^5\) service, that contains 310 terms, would be replicated to 310 bins. Some terms are very rare, like 'stockquote', while others are very common, like 'price'. It is reasonable to expect that a query for 'price' could be answered by many peers. Therefore, it would be a waste of resources and a cause of a network hot-spot to replicate the description to the peer responsible for this term (i.e. to the peer where the term 'price' maps to). On the other hand, 'stockquote' is rare, and the description should be replicated to the peer responsible for it, since it would be difficult to find another peer that has this rare term.

So let us assume that initially (by random choice) the peer responsible for 'stockquote' is selected for replication. It is very likely that this peer will already have descriptions with 'price' since (a) 'price' is quite a common term and (b) 'stockquote' and 'price' are correlated because they occur much more often together in a document than some random terms. Therefore, replicating to the peer responsible for 'price' should have a much lower probability than replicating to the peer responsible for 'stockquote', since it is likely that the peer responsible for 'stockquote' could also answer queries on 'price', while on the other hand queries on 'price' can be answered by many other peers (or at least by the peer responsible for 'price').

\(^4\)Although a purely term-based approach does not suffice to express queries based on the underlying semantics of WS descriptions, the decentralized indexing scheme that this work builds on can be used as a substrate that any query language can operate on top of

\(^5\)http://www.xignite.com/xQuotes.asmx?WSDL
5.3. OUR APPROACH

Algorithm 6 Rarity-based walk

Require: Let $\mathcal{D} \subseteq \{t_1, t_2, \ldots, t_n\}$ the set of all possible descriptions and let $d \in \mathcal{D}$ a description to be indexed. Let variable $T \subseteq d$, initially $\emptyset$, denote the set of terms that have been used in forwarding the description, parameter $f_{max}$ the frequency threshold for popular terms, $D_p \in \mathcal{D}$ the set of descriptions stored by peer $p$ and $D_{(p,t)} \in D_p$ the set of descriptions of peer $p$ that contain term $t$. Let $p_t$, the peer responsible for term $t$.

while true do
    if ($\exists t \in d | p_t = p$) then
        // Store item if peer is responsible for it
        $D := D \cup d$
    end if
    $t_{sel} := t_m | \forall t \in (d - T) |D(p,t)| \geq |D(p,t_m)|$
    $T := T \cup (t |D(p,t)| > f_{max}) \cup t_{sel}$
    if ($t_{sel} = \emptyset$) then
        return
    else
        // Send to the peer responsible for $t_{sel}$
        $p := p_{t_{sel}}$
    end if
end while

Require: A query for terms $q = \{t_1, \ldots, t_n\}$.

while true do
    $t_{sel} := t_m | \forall t \in (q - T), |D(p,t)| > |D(p,t_m)|$
    $T := T \cup t_{sel}$
    if ($t_{sel} = \emptyset \lor \text{querysatisfied}$) then
        return
    else
        //Send to the peer responsible for term
        $p := p_{t_{sel}}$
    end if
end while
Our algorithm is described in natural language in the following paragraphs and formally in Algorithm 6.

5.3.1 Inserting descriptions.

Insertion messages contain the description itself as well as a (initially empty) set of description terms that have already been used. All terms having frequency over a given threshold parameter $f_{\text{max}}$ in the local peer’s statistical information database are marked as used. The term that has the lowest frequency and has not been used is selected (i.e. the terms with the smallest number of occurrences in the descriptions stored locally in the peer). If such a term exists, it is marked as used and the message is forwarded to the peer responsible for that term in the DHT. The worst-case complexity of the inserting algorithm is when a description contains terms that are never used in another description before. In that case for each term $t$ in the description $D$ is routed to the ‘responsible’ peer via the DHT algorithm which means that the number of hops is the average number of DHT hops for a lookup times the number of terms, i.e. $|D| \times \log(N)$ where $N$ is the number of peers in the DHT overlay.

5.3.2 Querying.

For each term in the description, the hash-value is calculated and the query is routed to the peer in the DHT to which that value corresponds. However, if enough answers are found en-route by peers forwarding the message according to the DHT routing algorithm, then the message is not routed further to the destination peer and the query process for that term stops. This is meant to protect peers to which popular terms map to.

Compared to an algorithm that replicates according to terms chosen at random, our approach has negligible additional computational costs. This is because both determining rarest terms in a description and maintaining a list of term frequencies has a small computational overhead, since only calculations on the local peer are required.

An additional interesting property of the algorithm is its anytime behavior: in the beginning, when peers have no overview of which terms are rare, they will replicate descriptions to all peers since $f_{\text{max}}$ will not be reached. As the number of descriptions in the system grows, local term popularity knowledge in each peer will also grow, since peers can approximate the popularity of a term by counting the occurrences in their own data.

This mechanism suffices for determining adequacy of information, while it is also orthogonal to caching techniques or the use of shortcuts to peers, such as [115]. The worst-case complexity of the algorithm is in those cases when the number of answers in the system are less than the threshold for the routing algorithm to stop searching. Namely in that case, for each term $t$ in the query $Q$ a DHT lookup is made, meaning
that the number of hops in that case is $|Q| \times \log N$, where $N$ is the number of peers in the DHT overlay.

Although the queries currently are a list of keywords, also typed queries, or attribute-value pairs, can be handled in an efficient way. A simple method is to add the types to the keyword list in the query and let the rarity-based algorithm figure out by itself that the attribute is more popular then the value. For example the query

$$(\text{message\_name='tickerSymbol'}, \text{message\_element='string'}, \text{port\_name='StockQuotePort'}, \text{binding='StockQuoteBinding'})$$

would be translated to

$$(\text{message\_name, tickerSymbol, message\_element, binding, string, port\_name, StockQuotePort, StockQuoteBinding}).$$

### 5.4 Evaluation

In this Section we describe the setup for as well as the outcome of the empirical evaluation of our approach. This consists of an analysis of the creation process and properties of the seekda WSDL corpus, the definition of evaluation criteria and experimental settings, the implementation of our system, large-scale emulation using the DAS-3 distributed supercomputer and the analysis of the results.

#### 5.4.1 Dataset creation

Our experiments are based on real data provided by seekda, obtained during a focused crawl performed in March 2008.

Crawling considered at most one WSDL document per URI, even though the same WSDL might be retrieved from multiple URIs. This can happen, for example, if links differ in capitalization: “..?WSDL” vs. “..?wsdl”. WSDL well-formedness was verified using the wsdl4j\(^6\) Java API, but no further filtering was performed (such as eliminating duplicates or verifying the availability of the service implementation).

The next step after crawling was parsing the resulting WSDL data. Standard parsers (e.g. tsearch\(^7\)) are optimized for HTML web pages and only extract inline textual documentation for WSDL documents, while attribute names and values are ignored. The problem is further aggravated if we take into account the fact that only about 30% of WS have such explicit documentation\(^8\), without which a standard parser does not extract any terms at all.

---

\(^6\)http://sourceforge.net/projects/wsdl4j

\(^7\)http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

\(^8\)based on seekda’s database analysis
In order to increase the number of descriptive terms associated with a WSDL document, we developed and used our own specialized parser and tokenizer, that also takes into account the semantics of WSDL documents such as name attributes of service and portTypes. Attribute names in WSDL documents are very expressive and contain terms useful for indexing. To give an example of the extraction performance gain of our approach, we note that for the aforementioned stock quote example our parser has managed to extract 310 terms, compared to a mere 90 terms extracted by the tsearch2 parser.

Having collected all element names from the WSDL, we subsequently processed them considering naming conventions in programming, such as e.g. camel casing, a practice of writing compound words or phrases in which the words are joined without spaces and are capitalized within the compound. For example, for the service named “StockQuoteService”, our parser extracts the terms “Stock”, “Quote” and “Service”.

Additionally, the URL where the WSDL was found and all URLs mentioned within the WSDL (e.g. target namespace) were taken into account. Every URL was split into its components and then processed as described above. For example, from “http://aws-beta.amazon.com/onca/soap?Service=AWSProductData” we extract “http, aws, beta, amazon, com, onca, soap, service, product, data”. Note that we did not use any manually maintained stopword list, since our algorithm is capable of dealing with common terms.

### 5.4.2 Dataset properties

The resulting dataset can be described as follows:

**Corpus** We used 54,245 WSDL documents, each identified by a URL and associated with a set of terms. The distribution of the number of terms per document as a cumulative percentage is given in Fig. 5.1(c). 82% of all descriptions have less than 92 terms. Note that some descriptions have many terms: 61 descriptions have more than 1000 terms (not shown in the figure).

**Terms** On average, we extracted 76 terms per document. 251,436 unique terms appear a total of 4,151,141 times in the dataset. The distribution of term popularity is given in Fig. 5.1(a) (absolute term frequency by term appearances, expressed as cumulative percentage), and Fig. 5.1(b) (number of unique terms by term appearances). Fig. 5.1(a) shows that if we pick a random element from a description in the dataset, there is a 60% chance that it corresponds to a term that appears less than 1700 times in the dataset and 40% chance that it corresponds to a term that appears less than 400 times. Fig. 5.1(b) gives the distribution of the number of appearances of terms. For example, 63% of all terms appear only once in the dataset, 12% appear twice and 6% appear three times. The distribution is very steep as 81% of terms appear less than three times and one term appears 48,139 times (i.e. in 90% of the descriptions)

**Queries** To ensure that our experiments are realistic we used a sample of 1,414 distinct queries, retrieved from seekda’s anonymised query logs. Query term popularity
5.4. EVALUATION

Figure 5.1: Dataset statistics
distribution is displayed in Fig. 5.1(f), from which we can see that 70% of query terms appear only once. Fig. 5.1(e) that displays the number of terms per query shows that most queries contain a single term, while from the number of answers per query given in Fig. 5.1(d) we conclude that most queries have between 10 and 100 answers.

5.4.3 Evaluation criteria

We evaluated our system in terms of **description recall** and **load distribution**. To gain additional insight into **description recall**, we always took into consideration the number of answers in the system, but limited the number of answers we are interested in to a maximum of $N$. The reason is that in a discovery setting there is no point in trying to retrieve all answers - instead, we are interested in getting enough answers for the task at hand. This is not a limitation of our algorithm, but rather a choice to make our evaluation more realistic. For our experiments described in this chapter we made $N=25$ as a default value. Therefore, we define recall as follows:

$$D_{Recall} = \frac{|D_{relevant} \cap D_{relevant}|}{\min(|D_{relevant}|, 25)}$$

Furthermore, since we are doing exact string matching on conjunctive queries, the document precision of our system is one ($D_{Precision} = 1$).

The second criterion we will use for the evaluation of our system is the **load distribution** among peers. Although a number of techniques for achieving load balancing in DHTs exist, they all come at additional cost. Therefore, we aim to have a load distribution as close to uniform as possible.

5.4.4 System implementation

Our system is based on a three-layer architecture, in which the bottom layer is a DHT implementation, the middle layer consists of an object store and a distributed index supporting multi-attribute search and relies on the algorithms described in 5.3 and the top layer is application specific - in our case, WSDL search.

The system was implemented on Java 1.6. For the bottom layer, we have used the FreePastry DHT implementation, version 2.0\(^9\). The search primitives of the middle layer were provided by the implementation of the algorithm in Section 5.3. For the application layer, we simulated insertion of WSDL files and queries. Our experiments were based on a fully functional implementation, also integrated in the Openknowledge system [112].

\(^9\)http://www.freepastry.org
5.4.5 Large-scale emulation

We tested and evaluated our system on the DAS-3 distributed supercomputer\textsuperscript{10}, using 50 dual-processor dual-core nodes with 4GB of main memory each. Each node ran 8 processes (2 for each core). The head node of the local cluster of the DAS-3 acted as a bootstrap, being used as an access point to the system for the rest. We used Globus\textsuperscript{11} to start 400 instances of our system, which contacted the bootstrap node, and self-organized according to the Pastry DHT protocol. This process took less than 5 minutes.

Next, nodes published in parallel 136 descriptions each (52400 in total) at 400ms intervals. This amounted to a total publishing throughput of one description per millisecond. While publishing, 7.5\% of randomly chosen peers failed. The failure model was fail-silent (i.e. peers did not notify of their failure and stopped responding to messages).

Finally, each alive node made approximately 7 queries (for a total of 2754) and collected the results. Again, the queries were made at 400ms intervals, for a total querying throughput of one query per millisecond.

This publishing and querying interval was deliberately chosen so as to stress peers in order to test the quality of the implementation. Of course, this comes at the expense of recall, as we will see in 5.4.7. Results were written to files and processed off-line, so as to minimize the intrusiveness of the evaluation.

5.4.6 Experimental settings

Although we performed our experiments with a variety of settings, we made a selection of which ones to present in this evaluation based on their potential to give us insight on the performance of our algorithm. So we chose the following settings, that are on some level comparable to each other:

- **All**: We index the WSDL files by all terms in their descriptors. Theoretically, this should yield perfect recall. However, in order to index each description the required number of DHT messages and copies of the descriptions is equal to the number of terms (i.e. each description will be stored by as many peers as its terms).
- **Random**: Each description is indexed according to 10 terms, chosen randomly. This leads to a constant number of replicas and DHT messages at the cost of reduced recall. We have selected this number of terms because it leads to the same number of description messages as setting “Rarest-low”. Thus, we can perform a direct comparison.
- **Rarest-low**: Refers to the implementation of our rarity-based walk, with $f_{max} = 3$. It is to meant explore the lower bound where our system still yields good recall.
- **Rarest-high**: Refers to the implementation of our rarity-based walk, with $f_{max} = 20$. It is meant as a setting where we require recall comparable to setting “All”, but

\textsuperscript{10}http://www.cs.vu.nl/das3/
\textsuperscript{11}http://www.globus.org/
5.4.7 Results

Before presenting the outcome of our experiments, we should note that our results are not comparable to centralized indexing approaches or to fast indexing using parallel computing techniques: the focus of this work is to test our service discovery approach in a distributed setting, which can be extended to Wide Area Networks and scale to much larger numbers of nodes.

Table 5.1 shows our most important findings: Replicating indexes according to all
5.4. EVALUATION

terms\textsuperscript{12} gave comparable recall to the rarity-based walk with parameter $f_{\text{max}} = 20$ (setting rarest-high). Nevertheless, our approach used four times less messages, a significant improvement. In a scenario where less messages are used, we can see that our rarity-based walk with parameter $f_{\text{max}} = 3$ outperforms replication according to a set of terms chosen randomly, yielding 29\% higher recall with roughly the same number of messages.

Figure 5.2(a) gives us additional insight by plotting description recall by the total number of answers per query. The hypothesis that queries with many results are easily satisfied is confirmed, since for queries with more than 300 answers all approaches have almost perfect recall. The curves for all and random converge toward perfect recall, as expected. Note that the reasons behind the all setting not achieving perfect recall are heavy load and failing peers.

For queries with less than three answers, rarest-low and rarest-high perform equally well with all, using roughly seven and four times less messages for indexing, respectively. random performs significantly worse, yielding approximately four times less recall than the rest.

For queries with 3 to 50 results, rarest-high suffers a small decrease in recall while the recall of rarest-low is decreased to almost half. This does not happen with all and random, which see an increase. We attribute this to our algorithm using term popularity as a measure for description popularity. Of course, this is not always true: for example terms “banana” and “http” are popular, but a description containing the combination of these terms will not be common. Queries with very few answers usually contain one term that is almost unique in the descriptions indexed in the system. Our rarity-based approach will always index descriptions according to unique terms, thus recall is very high, contrary to random where recall is low. Queries with more terms are more likely to correspond to descriptions that contain more common terms which are not so likely to be indexed. This is the case where our rarity-based approach does not work optimally. Queries with many results are so easily satisfied that whether many of the terms of the corresponding descriptions are indexed or not does not play an important role.

In Fig. 5.2 (b) we can see how load is distributed among peers. The ideal curve would be a flat line, i.e. load distributed equally among peers. Our rarity-based approaches clearly outperform random and all as expected. Although a quantification would make more sense for a specific load-balancing mechanism, we note that the maximum number of messages received by any peer is approximately 30\% less in the rarity-based approaches. Standard deviation is also lower for the rarity-based approaches (see Table 5.1).

In total, even for a relatively small overlay of 400 peers, the benefits of our rarity-based approach are substantial. It is expected that as the network size grows, the load distribution problems for the random and all approaches will be aggravated.

\textsuperscript{12}note that the number of messages used is 66.97 instead of the theoretical 76 because of peer failures and lost messages
### Related work

A first group of systems we will examine are attempts to directly extend UDDI by applying some sort of distributed techniques on top of it.

In [88], a hierarchical P2P extension is proposed to UDDI, according to which peers can publish an advertisement about some WS they have to offer, or subscribe to be notified when some WS they are looking for becomes available. Peers are organised in syndications according to topics. Each syndication has its own UDDI registry and is served by a super-peer that propagates publications and subscriptions to a top-level UDDI registry, as well as being responsible for matching subscriptions and publications. Although this mechanism distributes the load of the top-level UDDI, therefore should in theory enhance scalability, it does not remove the SPoF vulnerability.

A similar hierarchical topology is Ad-UDDI [37]. Here we have three layers of UDDI servers - the root registry, the business service registry and the service layer. The business service registries are organised according to a predefined industry service classification scheme. In addition, there is also a mechanism for the active monitoring of registered services, in order to ensure that registry entries are not stale. Although simulations have shown improvement in performance, again the SPoF vulnerability remains.

Another group of systems tries to capitalize on the idea of organising a P2P network according to a globally known ontology in order to facilitate more efficient routing via topic clustering, presented in [104]. Here it is hinted that a P2P infrastructure based on such a topology (HyperCup) could be used to facilitate discovery of semantically annotated Web Services. Some proposals have been put forth that build on this, such as [87], which uses DAML-S and Gnutella as the ontology and P2P substrates correspondingly. A similar approach is [125], that is based on the WSMX execution environment for semantic Web services and also utilizes a HyperCup topology for the underlying P2P network of registries.

A system that utilizes both P2P and UDDI is Meteor-S [131]. Here peers hosting UDDI registries are classified based on their content according to a predefined global ontology and only host semantically annotated services that fall under their jurisdiction. Queries are submitted to the registry most relevant according to the classification of the desired service to be retrieved and the structure is overseen by

<table>
<thead>
<tr>
<th>Setting</th>
<th>Recall</th>
<th>#mess./descr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>94.67%</td>
<td>66.97</td>
</tr>
<tr>
<td>Rarest-high</td>
<td>93.45%</td>
<td>17.19</td>
</tr>
<tr>
<td>Rarest-low</td>
<td>74.04%</td>
<td>9.22</td>
</tr>
<tr>
<td>Random</td>
<td>57.21%</td>
<td>9.75</td>
</tr>
</tbody>
</table>

Table 5.1: Summary of experimental results
5.6. CONCLUSIONS AND FUTURE WORK

a unique gateway peer.

The systems mentioned above operate under two assumptions that limit their applicability for existing WS: the existence of a globally shared ontology/classification scheme, and the requirement for service annotations.

The group of systems closest to our own work is the one that utilizes DHT substrates in order to benefit from their efficiency and robustness.

The work presented in [105] uses Hilbert SFC to reduce the $d$-dimensional space of $d$ keywords used to annotate a WS to a 1-dimensional space that can be indexed on top of a DHT. In addition to the limitation of only being able to index on up to three terms for each WS, there is no reference to the dataset origin and characteristics in the system’s evaluation via simulation.

In [39], discovery is done on top of a DHT, but based on service behavior rather than annotation: service behavior is modeled as a Path Finite Automaton (PFA), and the representation of this model is hashed in order to store it in a DHT. Service retrieval is also based on formulating queries based on automata representations which are hashed and routed on the DHT node responsible for storing the corresponding service(s). Furthermore, there is a ranking mechanism based on reputation. It must be noted however that the system is not evaluated and the required PFA formalism for storage and retrieval is not supported by existing WS.

The Atlas system [78], although developed to deal with Semantic Grid resource discovery, is similar in principle to other DHT-based service discovery systems. It assumes service storage and retrieval are RDF(S)-based, using the RUL and RQL language correspondingly and a RDF(S) hashing scheme aimed at distributing the load efficiently over the DHT. It also supports a publish/subscribe mechanism, but this is the only part of the system that has been evaluated via simulation.

5.6 Conclusions and Future Work

In this chapter we presented a massively scalable WS discovery system built on P2P technology. Our contribution in the field lies in the following: Firstly we have crawled, parsed, extracted and analyzed the largest WS description corpus, to the best of our knowledge, with the benefit of having rich descriptions for WSDL files without the need for additional annotation. Secondly, we have developed an algorithm to index WS descriptions on a DHT using term popularity as a counter-indication for selectivity. Thirdly, we have implemented a system using the aforementioned algorithm and have performed extensive empirical evaluation using real data on an implementation running on a large number of peers. Finally, our approach showed significant performance gain, compared to two reference approaches.

Future work lies mainly in supporting a more expressive query language on top of our proof-of-concept term-based retrieval, as well in experimenting further in order to test how the gain in performance changes as the network size increases and measuring the robustness of our system against a higher peer churn rate. In addition,
even though our work focuses on WS discovery and not search, thus support for ranking results is not fundamental as it is for search engines, we intend to integrate ranking results in our system. We can achieve this by taking into account description scores, provided by an external system in the form of a scalar reputation value. To rank results, we can use a combination of their reputation value and their relevance to the issued queries.

5.7 Acknowledgements

This work is part of the OpenKnowledge project and the LarKC project, funded by the European 6th and 7th Framework Programme respectively.
Chapter 6

Privacy-preserving sharing

We have built on the approach described in chapter 3 to develop an architecture that offers scalable privacy. In this context, scalability refers both to the level of privacy provided and the size of the system.

With regard to the goals set in the introduction, this chapter pertains to scalability, interoperability, openness and control.

This chapter was published as “A scalable architecture for peer privacy on the Web” in the workshop “Trust and Privacy on the Social and Semantic Web” in the ESWC ’09.
abstract

We consider the privacy implications of exchanging Web data and present the design of a peer-to-peer discovery mechanism with strong privacy guarantees. Its benefits are that it does not rely on a trusted third party, spreads the computational cost among participants and places minimal restrictions on privacy policies. Furthermore, it provides scalability both in terms of system size and level of privacy offered. We outline an implementation that relies on two overlays: a distributed hash table for discovery and an anonymising overlay. Finally, we evaluate the proposed mechanism against a number of privacy threats and analyse its complexity.

6.1 Introduction

Organizations holding personal information (e.g. government, social networking websites, banks) provide a Web Service where user agents, after authenticating, can retrieve and modify information that concerns them. Users cannot keep track of the organizations with data that concerns them since the latter is sometimes not even inserted or maintained by them. How can these Web Services be located while maintaining the privacy of users and data? The basic challenges in designing such a system are that the number of such Web Services may be large, thus it is not possible to contact all of them for each query and that user agents and web services should only be able to see data and queries they are authorized to.

Centralized architectures where a single organization maintains an index of all data and enforces access control policies suffers from the following drawbacks: This organization must be completely trusted by all participants. Additionally, the maintenance of such a gargantuan volume of data would not be an easy, let alone financially viable, task. Finally, the centralized infrastructure should be able to enforce the access control and privacy policies of the information providers. This inhibits or precludes the use of locally calculated policies and policies that use private criteria. For example, some Web Service wants to authenticate users itself and does not share user passwords with the central index. This is an especially important restriction for the Social Web or any system where we do not have a priori trust agreements \cite{89}. For these reasons, we advocate the use of a peer-to-peer paradigm, where information providers will be able to keep data locally, or at least in the same authority domain and enforce their own privacy and access control policies.

The focus of this work is on scalable privacy, with scalable referring to both the
6.2. PRIVACY THREATS

The size of the system and the level of the privacy guarantees provided. The threat to the privacy of an individual is related to the ability of an adversary to associate information or other individuals with it. Instead of a one-size-fits-all approach, the level of privacy in our method is tunable. We explore the trade-off between privacy and performance ranging from high performance, providing similar performance and slightly better privacy guarantees than current resource discovery systems and search engines to high privacy, providing anonymity for the participating parties and non-disclosure of the content, its description or a unique identifier for it. Note that the focus of this work is not on privacy and access control policies, but on an architecture and mechanisms that allow a more open set of such policies.

We propose a Peer-to-Peer framework for private discovery and querying of (Semantic Web) data and develop a method to locate data providers without revealing their location, identity or the descriptions of the content. To this end we describe a distributed scalable index, implemented on top of a Distributed Hash Table (DHT). Additionally we show a method to obfuscate and approximate the descriptors in the index, so as to prevent disclosure of resource descriptions and combine it with an anonymising network to protect the identities of the parties involved. We test a selection of the possible privacy settings against a set of possible attacks and analyse their performance, paying special attention to the trade-off between the additional privacy mechanisms and the associated computational and network overhead.

In section 6.2 we describe the privacy threats covered by our work. Related work is described in section 6.3. In section 6.4 we introduce our architecture for scalable privacy, including a description of the technologies we employ to achieve this. Finally, we analyse our architecture in relation to a set of privacy attacks and the associations defined in section 6.2.

6.2 Privacy threats

We present the threats within a discovery/sharing system in terms of the ability of an adversary to associate aspects of content (e.g. a descriptor or the content itself) to a user. We use the term provider for a peer that shares content and seeker for a peer that consumes content. Content may be anything (e.g. RDF data or Social networking profile or a medical record) while a content identifier is a unique identifier for some content (e.g. a cryptographic hash of the content or a URI). A discovery mechanism facilitates the sharing process by allowing a seeker to either locate content or potential providers based on a set of descriptors (e.g. keywords or derivatives). The discovery mechanism usually maintains an index of descriptors to providers.

Table 6.1 presents possible privacy impairing association. We observe that: (a) Any association in the top cells is detrimental for privacy (e.g. a mapping from content to identity). (b) It is possible to derive more sensitive information from less sensitive information, moving from the bottom of the table to the top (e.g. an
CHAPTER 6. PRIVACY-PRESERVING SHARING

<table>
<thead>
<tr>
<th>What</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content (eg. document)</td>
<td>Identity (eg real name)</td>
</tr>
<tr>
<td>Content description (eg. keywords)</td>
<td></td>
</tr>
<tr>
<td>Query (eg. SPARQL query)</td>
<td>Location (eg. IP address)</td>
</tr>
<tr>
<td>Query description (eg. set of URIs)</td>
<td></td>
</tr>
<tr>
<td>Content ID (eg. GUID)</td>
<td>Any user in the system</td>
</tr>
<tr>
<td>System (usage of)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Privacy impairing associations. This table shows the different associations that can be made between content and identity ordered by descending significance.

(adversary that knows the set of URIs that appear in a SPARQL query can get a good estimate of the query itself). (c) Even perceived insignificant associations can lead to significant privacy breaches (e.g. two users that are associated with similar content IDs have similar interests, thus user profiling is possible). (d) different association types can be combined. For example, an adversary that knows the IP addresses where the system is accessed from (system - location association) and has access to an anonymous query log (query - any user in the system association). She can infer a (weak) association between queries and IP addresses. We will refer back to this table in section 6.5.

6.3 Background

Discovery/sharing systems are widely used both in closed domains and on the public web. Although encryption schemes exist to ensure security, significant challenges concerning privacy and scale remain. Search engines can scale but have abysmal privacy guarantees. Peer-to-peer systems satisfy the basic principle of lacking a central trusted party, but mechanisms to guarantee privacy are not available yet.

We provide an overview of the privacy implications (for seekers and providers) of current discovery/sharing mechanisms in combination with their scalability properties:

**Centralized index** An index of all content or content descriptions is kept at a collection of centrally managed servers. Information seekers post queries to the index and receive the content, a content identifier or a provider to ask for the content. This can be a scalable solution, assuming adequate financial resources to maintain this infrastructure. Nevertheless, it requires that all parties (both seekers and providers) trust this central index with their privacy, since it has complete control and access to all content and queries. Furthermore, providers should entrust the index to correctly enforce the policies specified with the data (if any).
6.3. BACKGROUND

Broadcast-based unstructured overlays Unstructured overlays are a type of peer-to-peer overlay where peers maintain a set of ad-hoc connections to other peers [58]. Queries are broadcasted to all peers, or a significant subset of peers in the system. They provide good privacy guarantees concerning the information providers, since there is no index that can be used to derive privacy sensitive information from. Additionally, every provider maintains full control of its own data. The privacy of the seekers on the other hand is not protected since their queries can be seen by any peer in the network. Broadcast-based systems are not scalable since queries need to be sent to a large subset of the total peers in the system.

Structured Overlays Structured peer-to-peer overlays impose a global structure on the peer connections [72]. They can be used to implement efficient and scalable indexes spread over a large number of nodes. They are very scalable but fare poorly with provider privacy since the index is readable by a large number of possibly untrusted nodes, which can be used to derive privacy sensitive information about the provider. Any single compromised index node is a direct threat to privacy of the seekers that uses it, since the queries are exposed.

Confidential indexes Confidential indexes aim at hiding information from the indexed content. The general pattern is to introduce noise to the results in order to avoid strong associations. We describe two recent implementations of confidential indexes that are very relevant to our work.

The confidential index proposed by Bawa et al [15] relies on a public index that divides the providers over a set of privacy groups. Within the group, they collaboratively create a bloom filter with the descriptors of their content using a randomized process so as to avoid exposure of the content of individual providers. These filters, along with the list of all providers in the group are sent to a central server, where an index is created that maps bits in the bloom filter to privacy groups. The privacy of the providers is protected since the privacy groups contain providers that have or do not have the requested content (introduction of false positives).

Zerber [134] uses a set of largely untrusted servers to maintain an inverted index of descriptors. Access control is enforced on the index. To defend against compromised index nodes, no single index server is given enough information to reconstruct a descriptor by itself. To this end, descriptors are encrypted using $k$ out of $n$ cryptography. This means that every descriptor is split and encrypted into $n$ different parts, each owned by a different index node. Since every index node enforces access control locally, as many as $k$ compromised index nodes are required for an adversary to obtain a complete descriptor. For additional protection, the index maps sets of terms instead of single terms to document descriptors (called a merged posting list mechanism). The de-
crypted descriptors contain the original index terms (so the irrelevant entries returned as result of the merging can be filtered out) and the location of the document. ZERBER+R [135] additionally provides top-k ranked results.

Our own method falls into this category and is further described in section 6.4.1. The major improvement over these systems is that our system additionally provides anonymity and can scale to a much larger index, since it is maintained by system participants.

**Encrypted indexes** Encrypted indexes protect the content by encrypting sensitive parts of the index entries. Since this generally requires complex key management schemes, scalability is a problem.

### 6.4 Our Architecture

We are using an index-assisted peer-to-peer model similar to the one in [15]. To have their information indexed, providers (a’) extract a content descriptor (e.g. keywords, hashes of keywords, ...) and (b’) store it in the index together with an identifier to contact the provider. To locate and access content, seekers need to take the following steps: (a) they create a set of content descriptors based on their query (e.g. a set of keywords, hashes of keywords, ...), (b) they send this set to the index which in turn (c) returns a number of providers which are (possibly) able to provide the information. (d) The seeker selects a number of these providers based on local preferences and (e) negotiates directly with the provider to retrieve the requested information. Our method places no restriction on the privacy and access control policies for step (e), except that providers need to be able to enforce them themselves locally. The focus of this work is on defining a scalable architecture for this model and develop mechanisms to guarantee privacy and anonymity.

#### 6.4.1 Scalable architecture

We will outline a distributed and scalable implementation. The meaning of the word scalable is two-fold: First, the performance of the system does not deteriorate severely as the number of participants increases. Second, different levels of privacy are supported. The infrastructure that will be presented functions through the synergy of two peer-to-peer overlays: the *Indexing overlay* and the *Anonymising Overlay*. The former provides a global scalable index on top of a DHT. The latter provides a distributed and scalable mechanism to hide the identity of peers, whenever this is required.

**Index-assisted query routing**

The *indexing overlay* is responsible for providing mappings from the set of descriptors in a query to a set providers that possibly have content that matches these
descriptors. It should be able to store a very large number of descriptors, thus a scalable implementation is required.

Distributed Hash Tables (DHTs) are a type of structured peer-to-peer overlays that impose a global structure on the peer connections [72]. Typically, each item stored in the DHT is associated with a hash ID chosen from a large key space. This space is partitioned in a way similar to hash tables, but instead of bins, they have peers. This distributed data-structure is self-maintaining, self-organizing and guarantees lookups and insertions using, in most cases, $O(\log(N))$ messages, where $N$ is the number of peers in the DHT overlay [73].

DHTs are well suited for implementing a large, global index, maintained by the participants of the system. This is, in fact, straightforward: for every element of the descriptor, peers insert in the index a $\langle$descriptor, peer-address$\rangle$ pair. For querying, seekers make one search for each of the query descriptors to retrieve the relevant providers.

The performance gain aside, using a DHT to maintain the index shifts the responsibility from one organization to multiple organisations, which is both a blessing and a curse: the index is no longer held in one location, meaning that no single entity has complete control of the entire index. On the other hand, the fact that it is partitioned means that many entities have control over parts of it. Obviously this has consequences for privacy.

Hiding the descriptors

Since we do not trust the index nodes, we cannot index the content descriptors in plaintext. We identify some methods to reduce the information conveyed in the indexed descriptors and make them non-understandable to prevent direct association of content and query descriptors to users or locations.

Obfuscation Instead of indexing the descriptor itself, we index its secure hash (e.g. using SHA-1) using hash function $h$. For example, for the descriptor $\langle$Soccer$\rangle = 134.23.32.2$, we will store $\langle h(\text{Soccer}) \rangle = 134.32.42.2$. When querying, we use the same hash function for the query descriptors. Note that secure hashes are a one-way function, so it is very difficult to retrieve the value that was hashed to produce a certain result.

Approximation It is possible to perform a dictionary attack by calculating the hashes of all descriptors the attacker is interested in and matching them to the indexed descriptors. To alleviate this problem, we introduce false positives in our results. Let $P(t)$ be the set of providers that actually have content with descriptor $t$ and $P'(t)$ the false positives. We define exposure for an index entry with descriptor $t$ as the ratio between the number of true positives relative to the total number of
answers $e_t = \frac{|P(t)|}{|P(t)| + |P'(t)|}$, a low $e$ meaning that the providers for $t$ have low exposure (high privacy) and $e = 1$ that they are fully exposed.

This is implemented by varying the size of the hash, so as to increase collisions. In previous work [65], we have shown that decreasing the length of the hash leads to a proportional increase in privacy. If we truncate hash values to $l$ bits, any query would match a portion of $r = \frac{1}{2^l}$ of all descriptors in the system. This is directly translatable to an average exposure of

$$e = \frac{1}{2^l} \quad (6.1)$$

For maximum privacy, $l$ will be equal to 0, i.e. any query would match all descriptors stored in the index and return all peers in the system. For maximum performance, $2^l$ should be much higher than the total number of descriptors in the system, so as to completely avoid collisions. In, chapter 3, we have shown that using an SHA-1 secure hash we can guarantee the privacy of individual terms for a typical webpage description corpus.

**Anonymising network**

Hiding the descriptors in the index is not enough: identities and locations can be used to perform association attacks (see table 6.1 and section 6.5). To provide stronger security guarantees, it is desirable to hide the identity of the peers in the network. Onion routing emerged from this general wish [26]. Onion routing anonymises communication channel and protects the identities and locations of the participants.

In onion routing, a set of nodes forms an anonymising routing overlay. The key principle, as described by Chaum[26], is that every router over which a packet is send is only aware of the identity of the previous router and the next router. This ensures that a single trusted router is in essence enough to protect the total path. This is achieved using layered encryption. Starting from the destination router, the peer will add the address of the router $A_n$ to the data, and encrypt it with the public key $K_n^{+}$ of the router before that in the routing chain. The resulting package is an onion containing several layers of encrypted addresses and data which at each level can only be read by the router with the correct private key:

$$\left[ A_1, \left[ A_2, \left[ P \right] K_2^{+} \right] K_1^{+} \right] K_0^{+}$$

When a router receives a packet, it will decrypt it with its own private key, and use the now readable address of the next router to send the contained (encrypted) data part to the next node. The destination will find the original payload when decrypting the last layer. The simplified scheme described here has been extended with nonces, symmetric keys and other mechanisms to properly ensure robust and scalable anonymity[36].
The original onion routing proposal defined 'return envelopes' to protect the identity of the client from the server. However, this method does not scale well. Next generation implementations provide other means to publish anonymously, e.g. the Hidden Services in the TOR project, which hide the identities both of the server and the client.

We can use this mechanism to provide volatile peer identities, i.e. pseudonyms. Instead of publishing \( \langle \text{descriptor, ip-address} \rangle \) pairs on the index, it is now possible to publish \( \langle \text{descriptor, pseudonym} \rangle \) pairs. After a seeker has obtained a psuedonym, it will use the anonymising network to contact the provider to retrieve the content anonymously. This will protect the privacy of the provider by making association of content to a real-world identity or location impossible.

In order to protect the descriptor from statistical correlation attacks, it is possible for a provider to publish using multiple pseudonyms. We will expand on this in section 6.5.

### 6.5 Analysis

Here we define four classes of settings based on different combinations of the techniques discussed above.

1. **Minimal** A DHT is used as an index. Keys are extracted from content and queries. They are published unmodified and associated with the (IP) address of the information provider. This setting protects associations with the content and the query, since only their descriptions are published or queried.

2. **Obfuscate and approximate** The key is obfuscated using a secure hash function. The level of privacy and the performance impact are inversely proportional to the length of the hash \( l \). This setting additionally protects query and content descriptors.

3. **Anonymise** The key is published unmodified but the identity of the information provider and the seeker is concealed using an anonymising network. Instead of a unique address, a pseudonym to contact the peer is published. The level of privacy can be chosen by the provider at indexing time, in the choice of how many terms are connected to a single service descriptor. Compared to setting 1, this setting additionally protects associations with identities or locations.

4. **Full** The key is obfuscated and an anonymising network is used. This setting offers the highest privacy guarantees. Both the level of obfuscation and anonymity can be chosen. This settings protects against all associations in table 6.1, except for the association that some unknown user is using the system and the association that the system is used by some location. Even though it
is not significant privacy-wise, even these last associations could be prevented by using a public anonymising network (e.g. [36])

**Attacks**

We analyse the susceptibility of the aforementioned approaches to a set of common attacks. We use table 6.1 as grounding to describe the consequences of successful attacks.

**A1: Compromised index** An adversary can seize control of the index. In our approach, this is possible by compromising the DHT (how this is possible is implementation-dependent and beyond the scope of this chapter). The privacy implications for a read-compromised index are negligible: the data in the index was already readable for everyone in the first place, thus no extra information is gained by taking over one or more index nodes. All settings are equally susceptible to compromising the DHT. For the rest of our analysis, we will assume that the DHT is read-compromised, i.e. that the full index is readable by the adversary.

**A2: Compromised anonymiser** Anonymity is provided only while the anonymising overlay is not compromised. Low-latency anonymisers are highly desirable for performance reasons, however they generally are susceptible to end-to-end communication correlation attacks [36]. Nevertheless, in order to perform this attack, the adversary needs to be able to eavesdrop on a large part of the network. Generally speaking, any compromised communication channel in setting 3 and 4 reduces the privacy for the peer(s) involved in that communication to the same level as in setting 1 and 2 respectively. The degradation of privacy is gradual and dependent on the power of the adversary to observe communication channels and the size of the system. Approaches exist to reduce these risks [36].

**A3: Taking the intersection of the returned providers for correlated descriptors** An attacker may exploit the fact that descriptors from the same provider are correlated. By posting several related queries and taking the intersection of the sets of returned providers, the attacker can associate provider locations or identities with descriptors.

For a set of $n$ queries with one descriptor $t_1...t_n$ each, the intersection of the sets of peers returned by the system is:

$$\bigcap_{i=1}^{n} P^*(t_i) = \bigcap_{i=1}^{n} (P'(t_i) \cup P(t_i)) = \bigg[P(t_1) \cap P(t_2) ... \cap P(t_n) \bigg] \cup \bigg[P'(t_1) \cap P'(t_2) ... \cap P'(t_n) \bigg] \cup ... \cup \left[\bigg[P(t_1) \cap P'(t_2) ... \cap P'(t_n) \bigg] \cup ... \cup \bigg[P'(t_1) \cap P(t_2) ... \cap P(t_n) \bigg] \right]$$

the set $P(t_1) \cap P(t_2) ... \cap P(t_n)$ represents all true positives for the intersection, while the $2^n - 1$ conjunctions represent the false positives, since each conjunction contains at least one $P'$. From previously given definition of exposure (formula 6.1),
6.5. ANALYSIS

we have that:
\[ e = \frac{|P(t_1) \cap P(t_2) \cap ... \cap P(t_n)|}{|P(t_1) \cap P(t_2) \cap ... \cap P(t_n)| \cup ... \cup |P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n)|} \]

We will consider the worst-case scenario where the adversary somehow knows that descriptors \( t_1 ... t_n \) are completely correlated in the index, i.e. they appear exactly on the same providers. Thus, \( P(t_1) = P(t_2) = ... = P(t_n) \). In this case, \( P(t_1) \cap P(t_2) \cap ... \cap P(t_n) = P(t_1) \)

\[ e = \frac{|P(t_1)|}{|P(t_1)| \cup |P(t_1) \cap P'(t_2)| \cup ... \cup |P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n)|} \]

In the denominator, all conjunctions with \( P(t_1) \) will be absorbed by the disjunction with \( P(t_1) \). Thus, \( e = \frac{|P(t_1)|}{|P(t_1) \cup |P(t_1) \cap P'(t_2)| \cup ... \cup |P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n)|} \). We also have that \( P(t) \cap P'(t) = 0 \), thus

\[ e = \frac{|P(t_1)|}{|P(t_1)| + |P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n)|} \]

The attacker has no control over the false positives \( P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n) \), which depend only on the properties of the dataset. Nevertheless, since \( A \supseteq A \cap B \), the size \( |P'(t_1) \cap P'(t_2) \cap ... \cap P'(t_n)| \) will decrease as the number of conjunctive terms increases. On the other hand, it becomes increasingly difficult to find a larger set of correlated descriptors.

Note that this attack is not possible in settings 3 and 4 since the providers are using pseudonyms. It is enough to publish each correlated descriptor under a different pseudonym to leave the adversary with no additional information about the set \( P(t_1) \cap P(t_2) \cap ... \cap P(t_n) \). If it is not possible for providers to know which terms are correlated, they can publish every descriptor under a different pseudonym.

The vulnerability of setting 2 depends on the length of the hash \( l \) (since the ratio of true positives \( P(t) \)) to false positives \( P'(t) \) is dependent on \( l \).

A4: Malicious provider tries to profile seekers A compromised index may collude with a provider to profile seekers based on their searches. Similar to the previously described attack, but profiling using queries instead of content thus attacking the privacy of the seeker instead of the provider. Seekers may prevent this by using several index nodes.

A5: Censorship and denial of service by index nodes Though strictly speaking not a privacy attack, censorship and denial of service are related attacks that can be prevented by providing peer privacy.

A malicious index node may censor all descriptors that correspond to a given descriptor, by not returning any answer to queries for those. In settings 1 and 3, this is easy for any subject, since the descriptors are stored in plaintext. In settings 2 and 4, this is more difficult: since multiple descriptors map to the same hash values, the
malicious node would have to censor all of them, since there is no way of knowing which ones refer to the given descriptor. The number of censored descriptors would be $\frac{1}{2^l} \cdot N$, where $l$ is the length of the hash and $N$ the total number of descriptors in the system. Detecting such a malicious index node is easy: it is enough to create some honeypot providers advertising descriptors that are likely to be censored and match results returned by the suspect index node.

Denying services to a given seeker or provider is possible in settings 1 and 2: a seeker could be presented with wrong or limited results, and the index could refuse storing the descriptors for a given provider. In settings 3 and 4, this is not possible, since providers use pseudonyms and seekers are anonymous. Assuming that malicious nodes also deny service to particular provider pseudonyms, the level of protection depends on the number of pseudonyms per provider.

**Performance**

We will analyse the performance of our architecture focusing on the trade-off between privacy and performance. Furthermore, we will focus on its scalability in terms of network size, index size and the throughput of the anonymiser.

In figure 6.1, we can see the communication channels in our system. Note that the circles represent *roles* fulfilled by a physical host. E.g. a single physical host may be part of the DHT, part of the anonymising network and a content provider. In fact, we will assume that the number of hosts participating in the DHT and the anonymiser increases linearly with the number of providers and seekers.

We have the following communication channels: *dd* and *aa* for communication among DHT nodes and anonymiser nodes respectively, *ad* for DHT nodes and anonymiser nodes, *sa* for seekers and anonymiser nodes and *pa* for providers and anonymiser nodes. The cost associated with sending one message through a communication channel is defined as $c_{ChannelName}$. E.g. the cost of doing a DHT search
is $c_{dd}$.

Considering that DHTs and anonymiser networks can have an arbitrary number of contact points (or entry nodes), channels $ad$, $pa$ and $sa$ will not be congested. The probable scalability issue may lie within the DHT network maintaining the index($dd$) or the anonymiser network($aa$).

The communication resulting from a query with $x$ descriptors is as follows: The seeker will divide it to $x$ sub-queries with one descriptor each. Each of these will have to be routed through the anonymiser to reach the DHT, where a lookup will take place. The DHT will return $s$ providers through the anonymiser, with a ratio $r$ of false positives. The protocol the seeker will use to negotiate with the providers is beyond the scope of our approach. We will assume that negotiation has a cost of $c_n$.

Thus, the cost associated with the query is:

$$c_q = x \cdot (c_{sa} + c_{aa} + c_{ad} + c_{dd} + c_{aa} + c_{sa}) + s \cdot c_n \cdot (c_{sa} + c_{pa})$$

A typical DHT lookup cost, in terms of IP messages is $O(c_{dd}) = O(\log(N_{DHT}))$, where $N_{DHT}$ is the number of nodes in the DHT and a typical anonymiser routing cost is $O(c_{aa}) = O(1)$. Sending messages over channels $sa$, $ad$ and $pa$ has a cost of one, since they are directly on top of the underlying network protocol. From these, we can bound the total cost of a query to:

$$O(c_q) = O(x \cdot (c_{sa} + c_{aa} + c_{ad} + c_{dd} + c_{aa} + c_{sa}) + s \cdot c_n \cdot (c_{sa} + c_{pa})) = x \cdot [2 \cdot O(1) + 2 \cdot O(1) + O(\log(N_{DHT}))] + 2 \cdot s \cdot c_n \cdot O(1) = O(x \cdot \log(N_{DHT}) + 2 \cdot s \cdot c_n)$$

From the definition of exposure and equation 6.1, we have that $e = a^s = \frac{1}{2^l}$, thus, $s = a \cdot 2^l$ where $a$ is the average number of providers that match a query(true positives). We consider the cost of the negotiation constant, thus $O(c_n) = O(1)$. Then, an upper bound for query cost is:

$$O(c_q) = O(x \cdot \log N_{DHT} + a \cdot 2^{l+1})$$

where $x$ is the number of descriptors in the query, $N_{DHT}$ is the number of nodes in the DHT, $a$ is the number of providers that match the query and $l$ is the length of the hash.

It is interesting to note that the cost for using the anonymiser does not increase the overall complexity for querying. Moreover, the size of the DHT, and thus the size of the index, increase only logarithmically, indicating good scalability. The size of the hash $l$ also plays an important role, signifying a trade-off between privacy and scalability. The limiting factor in our system is the number of matching peers $a$, since it is expected to grow linearly with the number of providers in the system. This makes the need for ranking in the index evident, but we will have to leave this for future work.

The cost of indexing consists of merely indexing every descriptor using the anonymiser. Due to space restrictions, we only present the final result:
\[ O(c_i) = O(x \cdot \log N_{DHT}) \]

where \( x \) is the number of descriptors and \( N_{DHT} \) the number of nodes in the DHT. This clearly scales well.

### 6.6 Conclusions

As pointed out in [89], privacy control has moved well beyond settings where an access control list or group access control would apply. Parties are no longer known in advance and privacy policies and trust have to be calculated on-the-fly. In an open (Semantic) Web, information sharing consists of locating and acquiring the data. Most privacy control approaches have focused on the latter. Our approach focuses on the former, providing an open discovery infrastructure where providers and queriers reveal minimal information. We have shown that our design is scalable in terms of system size and privacy level provided and that associations between aspects of content and identity can be protected. Furthermore, we have explored the trade-off between privacy and performance and analysed the message complexity for querying. A general recommendation about which setting to choose cannot be given, since it depends entirely on the requirements of an application and costs involved.

Future work includes implementation of this approach as a Sesame plug-in and integration of privacy policies.

### 6.7 Acknowledgements

We would like to thank Eyal Oren for the fruitful discussions and pointers and Frank van Harmelen for reviewing our work. This work is supported by the European Commission under the LarKC project (FP7-215535).
Chapter 7

Peer-to-peer backward reasoning

We propose a peer-to-peer model for reasoning where publishers retain control of their ontologies. Query resolution and reasoning is done in an incremental and distributed manner.

This work pertains to scalability and privacy and is based on the paper entitled “An architecture for peer-to-peer reasoning”, presented in the workshop “New forms of reasoning: dynamic, scalable and tolerant” in the ISWC ’07 [8].
abstract

Similar to the current Web, the key to realizing the Semantic Web is scale. Arguably, to achieve this, we need a good balance between participation cost and perceived benefit. The major obstacles lie in coping with large numbers of ontologies, authors and physical hosts, inconsistent or inaccurate statements and the large volume of instance data. Our focus is on scalability through distribution. Most current approaches split ontologies into triples and distribute them among peers participating in a structured peer-to-peer overlay. Identifying a series of drawbacks with this, we propose an alternative model where each peer maintains control of its ontologies.

7.1 Introduction

The success of the Web is attributed to its scalability and its low entry cost. One would expect at least the same requirements for the Semantic Web. Unfortunately, state-of-the-art technology permits for neither, as current methods and systems make assumptions that limit its usability, especially with regard to scale.

In [43], a series of assumptions in logical reasoning are identified, which are also largely present in infrastructures developed for the Semantic Web, namely Small set of axioms, e.g. limited number of concepts/relationships, Small number of facts, e.g. limited number of instance data, Trustworthiness, correctness, completeness and consistency of facts and axioms, implying some sort of central control or management and Static domains, e.g. infrequent updates or fixed semantics.

With aspirations toward a truly usable and global Semantic Web, research has turned into a number of directions such as approximation, trust infrastructures, database technologies and distribution. The focus of this chapter will be on distribution.

In this domain, peer-to-peer (p2p) systems are often seen as a vehicle for the democratization of distributed computing. Rather than relying in a possibly large set of professionally run commercial servers, they consist of community-volunteered hosts that collaborate on equal terms to achieve a common goal. Some of their perceived advantages are low cost, through the distribution of computation and self-organization, no single point of failure, due to their symmetric functionality and redundancy, no single point of administration or control, making censorship or preferential disclosure of information impossible and, under some conditions, scalability, due to the fact that the network can grow on demand.
We can tap into the vast resources offered by p2p systems to develop scalable infrastructures for the Semantic Web. A plethora of approaches has already been suggested [22, 2, 129, 77, 70, 13, 120, 50, 80, 57], mainly focusing how to efficiently distribute large numbers of triples among peers in the network. We argue against this approach, claiming that although it solves scalability issues concerning the number of facts in the system, it fails to address the rest of the issues mentioned above and, in some cases, it actually makes additional non-realistic assumptions.

We propose an alternative approach, using ontologies instead of triples as the standard level of data granularity, thus moving complexity from the p2p overlay to peer interactions. This allows for efficient and secure maintenance of information provenance and control of the publishers over access and availability of information. We also hope that this model will eventually facilitate the development of methods to attest results calculated in a distributed manner and improve performance over current systems, since it can exploit concept locality in ontologies.

We are aspiring to combine the scalability of structured p2p overlays with the perceived advantages of our model. To this end, we are sketching an architecture that uses a global index maintained by a Distributed Hash Table (DHT) to find the correct peers that interact to resolve queries. Furthermore, some technologies that would be useful in this architecture are suggested.

The rest of the chapter is structured as follows: In section 7.2.2 we are presenting the most important systems for RDF (Resource Description Language) storage and reasoning. We argue that there is a need for a shift of paradigm in section 7.3. Section 7.4 is a description of our approach, for which we are giving some performance indicators in section 7.5. We are concluding and outlining future work in section 7.6.

7.2 Relevant literature

7.2.1 Distributed hash tables

DHTs are a well researched flavour of structured p2p systems [73]. Nodes function autonomously and collectively form a complete and efficient system without any central coordination. In DHT overlays, each object is associated with a key, chosen from a large space. This space is partitioned in zones, and each peer is responsible for the keys and corresponding objects in a zone. Peers need to maintain connections only to a limited number of other peers and the overlay has the ability to self-organize, with respect to peer connections and object distribution, to handle network churn. In principle, all DHT-based systems provide the following functionality: store(key, object) storing an object identified by its key, and search(key) which returns the object (when it exists) from the peer responsible for the key. Current systems need approximately $O(\log(N))$ messages to search or store and each peer needs to keep from $O(1)$ to $O(\log(N))$ pointers to other peers, where $N$ is the number of peers in the network.
7.2.2 Scalable RDF storage

DHT-based

Research into scalable RDF storage lies closest to the focus of this chapter. Considerable research has been conducted in the area with most approaches sharing the following fundamental design choices:

- RDF queries are broken down into subqueries, namely triples with one or more variable values, for instance `<?, ns:lives_in, cities:amsterdam>`.`
- Query results are sets of bindings for variables.
- No single node can be assumed to have the answers to all subqueries, so the problem then consists of decomposing the original query and routing the ‘right’ subqueries to the ‘right’ node, and then composing partial results to obtain the answer to the original query.

The first to propose the use of DHTs to implement a distributed RDF store was RDFPeers [22]. The basic functionality for storing RDF triples involves hashing the triple’s subject, predicate and object and storing it in the three peers that are responsible for each of the resulting keys. Queries are answered by hashing the (at least one) constant part of the query triple pattern and routing the query to the node responsible for storing that constant. RDFPeers has the ability to resolve atomic, disjunctive and conjunctive multi-predicate RDF queries at a minimum cost of \( \log(N) \) (for atomic queries), however it has poor load balancing capabilities, completely lacks reasoning support and assumes a shared RDF schema.

GridVine [2] constitutes a logical layer of services offered on top of the P-Grid [1] DHT. It exposes higher level functionalities built on top of P-Grid: Insert(RDF schema), Insert(RDF triple), Insert(Schema translation) and SearchFor(query). RDF triples are inserted into GridVine by using the same method introduced by RDFPeers and it can also answer the same set of queries, but has the additional advantage of supporting translations between RDF schemata.

PAGE [129] is a proposal for a distributed RDF repository implementation that combines the index structure of YARS [52] with a DHT. YARS uses 6 different indexes and stores RDF quads (triples augmented with context information). PAGE works by using the same indexes (hence replicating triples 6 times) and achieves more efficient query processing, but also lacks reasoning support and load balancing.

RDFCube [77] builds on RDFPeers by adding a second overlay that indexes triples based on an ‘existence bit’ and then performs a logical AND operation on this existence bit before actually retrieving the triples when evaluating a query. This results in more lookups and higher maintenance cost for the extra overlay, but reduces the required amount of data that has to be transferred on the network.

[70] proposes two different algorithms for evaluating conjunctive multi-predicate queries. The first one, QC, uses the indexing scheme of RDFPeers to index triples,
with a small modification: if there are more than one constant parts for a subquery, then preference is given to indexing on the subject, then the object and then the predicate, as it is expected that this will also be their ranking according to discrete values. Subqueries are also sorted according to expected selectivity before execution. Then, a ‘query chain’ is formed that consists of the nodes responsible for each subquery. The second algorithm, SBV, uses additional triple indexing and dynamic query chain formation exploiting local variable bindings for subqueries.

In BabelPeers [13], nodes are also organized in a DHT overlay, and inserted triples are hashed on their subject, predicate and object, and stored by the node responsible for the resulting key. BabelPeers nodes however host different RDF repositories, making a distinction between local and incoming knowledge and applying RDFS reasoning rules. Nodes are additionally organized in a tree overlay structure in order to deal with overly popular values.

Non-DHT based

[120] is based on the notion of path queries to build an index only on paths and subpaths, but not on individual elements for a datasource. Every RDF model is seen as a graph, where nodes correspond to resources and arcs to properties linking these resources. The result of a query to such a model is a set of subgraphs corresponding to a path expression. Since the information that makes up a path might be distributed across different datasources, the index structure to use should also contain information about subpaths without losing the advantage of indexing complete paths, and the most suitable way to represent this index structure is a hierarchy, where the source index of the indexed path is the root element. In terms of space, the complexity of the index is $O( s \times n^2)$, where $s$ is the number of sources and $n$ is the length of the schema path. The trade-off is that query answering without index support at the instance level is much more computationally intensive, so different techniques (partly similar to the ones used in [70], in terms of query chain formation and subquery ordering) are applied on the basis of an initial naive query-processing algorithm in order to perform join ordering and overall optimization, under the assumption that nodes do not have local join capabilities.

Bibster [50] follows an unstructured semantic-based p2p architecture: each peer knows about its expertise and finds out about the expertise of neighboring peers through active advertising. Thus peers for expertise clusters. When a peer receives a query, it tries to answer it, or forwards it to other peers whom it judges likely to be able to answer the query, based on similarity functions between the subject of the query and the previously-advertised expertise topics, using the schema hierarchy and text-similarity methods.

Edutella [80] is a p2p architecture designed for distributed search of educational content based on meta-data. The meta-data is stored in RDF format in distributed repositories that form a super-peer-based p2p network, arranged in a hypercube topology. While it allows the use of multiple schemas, neither mapping nor RDF
semantics are supported. Additionally it uses a broadcast-based approach which would not scale gracefully.

Federated RDF repositories [57] aim at offering unified access among different RDF repositories by integrating them according to the federated repositories approach. Semantic Federations are collections of heterogeneous distributed RDF repositories that can be accessed as a unique local Semantic Repository. This approach however is based on static definition of participating repositories and uses flooding to distribute queries among repositories; it therefore lacks the ability to scale and to dynamically update federation membership.

7.3 Motivation

For the remainder of this chapter, we will focus on systems using a DHT infrastructure, since, so far, they are the only scalable solutions that do not rely on fixed schemata. Efficient as they may be in storing instance data and ontologies, these approaches do not address scalability in reasoning, are not dealing with provenance of information and do not support user/peer control over their own data. Hence, we argue that they are not appropriate infrastructures for the Semantic Web and are more similar to distributed databases, useful and important in their own regard. In the following paragraphs, we highlight some of their shortcomings, also in respect to the set of criteria mentioned in the introduction.

7.3.1 Reasoning

Partly due to their computational complexity, current reasoning techniques do not scale beyond a relatively small set of axioms. Focusing on approaches that distribute the reasoning process and, in particular, some of the systems presented in section 7.2.2, we can identify performance problems in both storing and retrieving triples:

Storing All approaches that support reasoning store the transitive closure of triples. Assume a music hierarchy where a class “Music” has hundreds of subclasses like “Rock”, “Pop” etc. Storing the statement $<$Joe, likes, 70’s Rock$>$ implies storing a triple for each superclass of rock (e.g. $<$Joe, likes, Classic Rock$>$, $<$Joe, likes, Rock$>$, $<$Joe, likes, Music$>$ etc), which may count in the dozens. Similarly, assuming that we use an approach like [2], to store these tuples in a DHT, we will need at least twice the number of messages as the number of tuples to be stored. To make matters worse, updating the ontology can be very expensive. Adding the statement $<$Music, subclass_of, Art$>$ means that for all statements with Music, we need to insert an additional triple. The number of these triples increases by $O(N)$ with the number of axioms in the system, i.e. we have overall storage and message complexity of $O(M \times N)$ where M is the number of facts and N is the number of axioms in the system.
7.3. **MOTIVATION**

**Querying**  Let us assume a query to find all subclasses of **Music** which are not a subclass of **Rock**. Resolving this implies retrieving all triples `<?,subclass_of,Rock>` and proceeding recursively down the hierarchy. Then all subclasses of **Music** have to be retrieved and the intersection of the two sets has to be calculated. To resolve this query, the entire hierarchy has to be retrieved. Although in terms of data traffic, this may sometimes be acceptable, the number of messages required is prohibitively high: resolving this query means sending at minimum a number of DHT messages roughly equal to the number of concepts in the hierarchy.

*The aforementioned examples clearly indicate the shortcomings in the current approaches for triple generation in large-scale systems.*

### 7.3.2 Control over ontologies

All DHT-based stores presented in section 7.2.2 share the following design assumption: *All ontologies and instance data are made public and are maintained in a distributed manner*. This is done by using the triple notation and distributing these triples among the hosts in the network, according to some indexing scheme. This means that hosts effectively have no control over the location and administration of their ontologies and instance data. We can identify the following weaknesses in this design:

**Provenance of information** The issue of information reliability that pertains the Web is also valid and even more exacerbated for the Semantic Web, since in this case information is meant to be processed and acted upon via automated reasoning techniques. Existing techniques[24] dealing with this issue are limiting in that they do not enforce identity verification, but assume a trusted environment. On the other hand, the only way to guarantee data integrity in such a distributed and dynamic environment would be the use of electronic signatures; i.e. each peer signs the triples it inserts in the system using its electronic key, which is certified by some certification authority. This however would impose a disproportionate overhead, since storing an electronic signature for each triple would require more space than the triple itself.

**Publishers are not in control of their ontologies** Ontologies and instance data are becoming important assets for businesses and organizations as they are expensive to develop, may contain business intelligence etc. Thus, it is very unlikely that publishers would want to relinquish their control to a set of community-volunteered computers. This would be as preposterous as suggesting to large companies to use one of the existing p2p file sharing systems to distribute their software.

**Ontologies and instance data are made public** Even in the case where relinquishing control would be acceptable, there would be many cases where access
control would be required. Again important issues arise on how should this access control be implemented by a number of untrusted and unreliable peers.

Having identified a set of limitations that could inhibit the development of the Semantic Web on current infrastructures, we will propose an alternative paradigm that could provide solutions to some of these problems and lay the foundation for future research.

7.4 Our approach

The main innovation of our approach is shifting the level of granularity for peer data from triples to entire ontologies. We propose a model where peers retain control of their ontologies and reasoning is done in a p2p manner. Query resolution is done in an incremental and distributed manner.

All peers have reasoning capabilities and are able to decide when they have had enough answers and query processing should be finished. Furthermore, queries can be decomposed into triple patterns (e.g. `<?, type_of, mtv: MUSIC>`). Figure 7.1 summarizes our proposed model. We will illustrate the explanation of each step using a simple example, the resolution of the query

\[ \text{SELECT } X \text{ WHERE } X \text{ type_of mtv: music} \]

using RDFS reasoning rules (i.e. this query should return all X that are the predicate of a “type_of” relationship with object being mtv: music or any of its subclasses.)

Figure 7.1: Querying in our proposed model.
1. **Partition query and select sub-query** Initially, the part of the query to be resolved first needs to be determined. Our example query can be written in a triple pattern format as $?t, type_of, mtv:music$. Obviously, there is no point in splitting this query further.

2. **Determine concepts and relationships required for reasoning** Out of the triple pattern $?t, type_of, mtv:music$, we need to select a starting point for routing our query. There are the following two choices: `type_of` and `mtv:music`. Intuitively, the best choice would be `mtv:music`, since it is more selective and we can use semantic routing techniques to determine that in a distributed manner. Furthermore, note that instead of `mtv:music`, we may have a literal and not a concept. In this case, we will need to anchor it to a concept. This is where ontology anchoring and ontology mapping techniques come in handy.

3. **Localize concepts and relationships required for reasoning** For this step, either the triples that match the pattern have to be retrieved or the query should be forwarded to the peer(s) that store the ontology(-ies) with these triples. In our example, as in most cases, it is wiser to forward the query, since it is much smaller in size (just a single pattern with no results so far, in this case).

4. **Perform reasoning locally** Now reasoning can be performed locally and the first results can be returned.

5. **Determine if answers are adequate** The next choice is whether the retrieved results were enough for the user or application. If enough results were found, query resolution stops, otherwise, the query is reformulated to be further processed.

6. **Reformulate query** To retrieve additional results and according to RDFS semantics, instances that have a type which is a subclass of `mtv:music` should be returned\(^1\). Therefore, we can reformulate the query as follows:

   SELECT X WHERE X type_of Y and Y subclass_of mtv:music

\(^1\) Note that this is not the only RDFS rule that applies in this case; for instance, we could look for subclasses of the relationship

1'. Now, the query will be SELECT X WHERE X type_of Y and Y subclass_of mtv:music. The choice now lies between pattern $?t, type_of, $?> and $?t, subclass_of, mtv:music>. The latter is preferred, since it has more bounded variables.

2'. `mtv:music` will be preferred over `subclass_of`, since it is more selective.
3’. The local peer is already knowledgeable about mtv:music (see 3.), so chances are, no forwarding is needed.

4’. The Y that are subclasses of music are found.

5’. Answers are still not adequate.

6’. Query is reformulated as `SELECT X WHERE X type_of Z and Z subclass_of Y`

1”’. `<?, subclass_of, Y>` will be selected.

2”’. Y will be selected.

3”’. Query will be forwarded to peers with some of the possible Y.

4”’. Additional results will be returned

5”’. Assuming that there are now enough answers, querying is finished. Otherwise, we can continue with step 6.

7.4.1 Architecture

We propose an architecture abiding to the above model. Ontology descriptions or part of ontologies (i.e. concepts or relationships) are stored in a distributed public index and querying takes place in a p2p manner. The public index is maintained by a DHT consisting of a set of volunteer peers with adequate computational resources and fast, stable Internet connections. This index is used to resolve URIs to locations, i.e. locating the peers containing the ontologies and instance data for each relationship, concept or instance and to Anchor terms to concepts in ontologies, in case we want to anchor literals to ontology concepts or relationships (e.g. anchor “lives” to `namespace1:lives`).

Each peer stores a number of ontologies. Although ontologies may be moved across peers and replicated, this is not necessary. i.e. peers may choose to retain complete control over their ontologies or replicate them for performance. For instance, the RDFS ontology is used in the inference process and is public. So, it should be replicated to practically all peers for performance reasons. On the other hand, some peers may decide that they do not want their ontologies fully disclosed, and therefore store them only locally and answer queries on them. For such cases, the approach described in chapter 3 and chapter 6 comes to direct use.

In the simplest form of the system, all URI lookups are done through the DHT. Indexing is equally straightforward: Peers store on the DHT mappings from the URIs of the resources they want to answer to their address.
7.4.2 Optimizations

A series of simple optimizations are suggested to improve the efficiency of the system.

**Triple caches** To avoid redundant network messages, peers may cache received triples. This would drastically improve performance but it would also imply some sort of soft-state mechanism to manage updates or deletions.

**Ontology caches/replicas** Sometimes, a peer may need data from an ontology so often, that it would make sense to keep a copy of the entire ontology, and perhaps share it with other peers. Note however, that this would only be possible for public ontologies.

**A semantic topology** Apart from maintaining the global index, peers can be organized in a semantic topology, determined by the overlap of their resource descriptions. To this end, they would maintain a set of pointers to “interesting” peers, along with the resource descriptions they contain. This would substitute expensive DHT messages with direct network messages and would improve performance on the expense of some additional storage space per peer, which is generally considered of minor importance. Updating these pointers is straightforward. When a new ontology is inserted with a triple \(<X,r,Y>\). For each triple, a pointer will be stored to the peer with the relevant concepts/relationships. For example, for \(<\text{wwf:seal}, \text{rdfs:subclass_of}, \text{mom:monk_seal}>\), we will make a lookup for \text{wwf:seal} and \text{mom:monk_seal} and retrieve the peers which have triples with these concepts. We will store a pointer to the publishing peer to each one of those peers. Thus, future queries that involve these concepts will be forwarded without having to consult the DHT.

7.5 Performance indicators

In this section we try to evaluate how our system would work by analyzing some properties of ontologies currently available on the web. For example, by analyzing the re-use of concepts (i.e. inter-linkage) between ontologies we can predict the consequences on how scalable our approach is since our approach performs better where there is not much re-use.

Swoogle [35] is a search and meta-data engine for the Semantic Web. Besides the core search functionality, it also provides detailed statistics about the more than 10,000 ontologies it stores, where Swoogle considers a *Semantic Web document* (SWD) to be a document represented as an RDF graph and a *term* refers to a rdfs:Resource node in a SWD.
CHAPTER 7. PEER-TO-PEER BACKWARD REASONING

Figure 7.2: The proposed architecture. 1' Peers index (parts of) their ontologies by sending a flat list to the distributed index. 1-6 Querying consists of (1) Lookup on the index for a peer containing concepts or relationships that are part of the query, (2) Index returns the address(es) of the matching peer(s), (3) Query is forwarded to the selected peer(s), (4) Peer 1 creates a new sub-query according to RDFS reasoning rules and forwards it to Peer 2 using the semantic topology, (5) Peer 2 returns the results of the subquery to Peer 1, (6) Peer 1 aggregates the results and returns them to the querying peer.
7.6. CONCLUSIONS AND FUTURE WORK

7.5.1 Namespace usage
Namespaces used in an ontology are pointers to other ontologies and therefore an indication of re-use. Their statistics\(^2\) show that there are 4576 namespaces used by 329987 SWDs. The purple line in figure 7.3a\(^3\) shows the distribution of namespaces. As can be seen, the popularity follows some power-law distribution, meaning that only a few namespaces are very popular (like the \texttt{rdf} namespace) and most are rarely used. This confirms our hypothesis that ontologies are not strongly connected which means that in most cases the possible answers can be found locally on a single peer hosting the ontology/-ies of interest.

7.5.2 Local reasoning
Swoogle provides statistics on the distribution of SWDs per website. There are 132206 websites indexed that are hosting 337182 SWDs, meaning an average of three SWDs per website. However figure 7.3a shows that the distribution follows Zipf’s law except in the tail, meaning that most hosts\(^4\) will only have one or two ontologies. If we combine this fact with the statistics on the number of terms per SWD (distribution shown in figure 7.3b) we see that in most cases local reasoning only needs to be done over a relatively small ontologies. Namely, the figure shows that the number of class and property definitions in most cases is smaller than 10 is one order of magnitude smaller than the number of populations. Most SWDs do not define classes or properties at all, but just populate instances, meaning that in most cases only local reasoning on instance checking needs to be done.

The number of SWDs per suffix\(^5\), not shown here, shows that most ontologies are written only in rdf and only a few also in owl, daml or rdfs, meaning that not much extra reasoning is needed currently than simple rdf triple matching. For now, this is a counter argument to our approach in favor to distributed triple storage mechanisms in terms of the need of lack of complex local reasoning in the latter approach. However the arguments of desired local control and provenance still favor our approach. Besides this, [34] states that the increased use of two OWL equality assertions: \texttt{owl:sameAs} (279,648 assertions in 17,425 SWDs) and \texttt{owl:equivalentClass} (69,681 assertions in 4,341 SWDs) may be an indication of increased ontology alignment, and therefore increased use of richer languages.

7.6 Conclusions and future work
In this chapter, we have presented a new method for distributed ontology storage and querying which has ontologies as the normal level of granularity for data distribution.


\(^3\)http://swoogle.umbc.edu/2005/modules/Swoogle\_Statistics/images/figure5-2004-09.png

\(^4\)note that we consider the number of hosts to be equal to the number of websites

Figure 7.3: **top:** Cumulative Term/Namespace usage Distribution, **bottom:** Cumulative SWD
Examining the ontologies currently on the Internet indicates that local reasoning is, most of the times, sufficient for query resolution. In this case, our approach clearly outperforms ones that rely on triple distribution on top of DHT.

Future work lies in more diligent evaluation of our approach, doing simulation and emulation experiments. Furthermore, we have not examined the scenario where peers do not have the capacity to store their own ontologies/instance data. In this case, the latter would have to be split and distributed among several peers. It would be very interesting to investigate methods to accomplish that, for example using past queries to determine which concepts/instances/relationships are used together, or splitting the ontology graph so as to keep overlap between the resulting graphs to a minimum.
Chapter 8

Distributed forward reasoning

Some forms of reasoning with Semantic Web data can be tackled as a discovery problem. Namely, how can we find triples that lead to inferences. We apply the method described in chapter 4, to perform distributed reasoning on large scale.

Semantic Web reasoning is closely related to interoperability. Furthermore, the process is guaranteed to eventually complete. Finally, this method can scale to large number of nodes and large datasets.

This chapter is based on work on the MaRVIN platform, an initial version of which won the 3rd place in the 1st Billion Triple Challenge at ISWC ’08. The text is based on “MARVIN: A platform for large-scale analysis of Semantic Web data” which was presented as a poster in WebSci ’09 [84] and on “Marvin: distributed reasoning over large-scale Semantic Web data” to appear in the special issue The Semantic Web Challenge 2008 of the Journal of Web Semantics.
CHAPTER 8. DISTRIBUTED FORWARD REASONING

abstract

Many Semantic Web problems are difficult to solve through common divide-and-conquer strategies, since they are hard to partition. We present MARVIN, a parallel and distributed platform for processing large amounts of RDF data, on a network of loosely-coupled peers. We present our divide-conquer-swap strategy and show that this model converges towards completeness.

Within this strategy, we address the problem of making distributed reasoning scalable and load-balanced. We use the Speed-Date algorithm, presented in chapter 4, which combines data clustering with random exchanges. The random exchanges ensure load balancing, while the data clustering attempts to maximise efficiency. We evaluate our overall MARVIN system for performance, scalability, load balancing and efficiency.

8.1 Introduction

Over the recent years, large volumes of Semantic Web data have become available, to the extent that the data is quickly outgrowing the capacity of storage systems and reasoning engines. Through the “linking open data” initiative, and through crawling and indexing infrastructures [82], datasets with millions or billions of triples are now readily available. These datasets contain RDF triples and many RDFS and OWL statements with implicit semantics [33].

Since the datasets involved are typically very large, efficient techniques are needed for scalable execution of analysis jobs over these datasets. Traditionally, scaling computation through a divide-and-conquer strategy has been successful in a wide range of data analysis settings. Dedicated techniques have been developed for analysis of Web-scale data through a divide-and-conquer strategy, such as MapReduce [32].

In contrast to other analysis tasks concerning Web data, it is not clear how to solve many Semantic Web problems (e.g. reasoning and querying) through divide-and-conquer, since it is hard to split the problem into independent partitions. However, to process, analyse, and interpret such datasets collected from the Web, infrastructure is needed that can scale to these sizes, and can exploit the semantics in these datasets.

To illustrate this problem we will focus on a common and typical problem: computing the deductive closure of these datasets through logical reasoning. Recent benchmarks [48, 17] show that current RDF stores can barely scale to the current volumes of data, even without this kind of logical reasoning.
To deal with massive volumes of Semantic Web data, we aim at building RDF engines that offer \textit{massively scalable} reasoning. In our opinion, such scalability can be achieved by combining the following approaches:

- using \textit{high-performance and distributed computing} infrastructure which run \textit{parallel and distributed algorithms} that exploit hardware varying from tens to many hundreds of processors.

- designing \textit{anytime algorithms} that produce \textit{sound results} where the degree of completeness increases over time.

- our novel \textit{divide-conquer-swap} strategy, which extends the traditional approach of divide-and-conquer with an iterative procedure whose result converges towards completeness over time.

We have implemented our approach in \textsc{Marvin}$^1$ (MAssive RDF Versatile Inference Network), a parallel and distributed platform for processing large amounts of RDF data. \textsc{Marvin} consists of a network of loosely-coupled machines using a peer-to-peer model and does not require splitting the problem in independent subparts. \textsc{Marvin} is based on the approach of \textit{divide-conquer-swap}: peers autonomously partition the problem in some manner, each operate on some subproblem to find partial solutions, and then re-partition their part and swap it with another peer; all peers keep re-partitioning, solving, and swapping to find all solutions.

In this chapter, we present our general approach called \textit{divide-conquer-swap} and show that the model is sound, converges, and reaches completeness eventually. We then focus on efficient computation: we introduce a distributed \textit{exit-door policy} for handling produced duplicates, and introduce our \textsc{SpeedDate} approach that combines efficient deductions while balancing the load equally amongst computation nodes. We report on simulation results with our \textsc{SpeedDate} approach and provide experimental results using \textsc{Marvin} on RDF graphs.

8.2 Related work

In general, logical reasoning allows us to expand RDF graphs with implicit information. For example, by combining \texttt{(Amsterdam locatedIn Netherlands)} and \texttt{(Netherlands locatedIn Europe)}, we can derive \texttt{(Amsterdam locatedIn Europe)}. Given the size of the data, scalable reasoning is a major challenge \[106, 44\]

8.2.1 Distributed reasoning

For distributed reasoning, triples that share common elements (“Amsterdam” in the example above) should be co-located at the same machine and combined into

$^1$named after Marvin, the paranoid android from the Hitchhiker’s Guide to the Galaxy. Marvin has “a brain the size of a planet” which he can seldomly use: the true horror of Marvin’s existence is that no task would occupy even the tiniest fraction of his vast intellect.
additional triples: triples should “meet” each other in one of the distributed peers. The challenge in this scenario lies in assigning rendezvous points for triples.

Our baseline approach uses random rendezvous points: triples are sent around randomly until they happen to produce some deduction. This random approach is load-balanced (all nodes hold the same number of triples at any point in time) but inefficient and not scalable: with a growing number of nodes, triples have less chance to meet.

Several distributed reasoning techniques have been proposed based on deterministic rendezvous points using a distributed hashtable (DHTs) [72]. Here, each triple is sent to three rendezvous peers (one for each of its terms: subject, predicate, and object), which ensures that triples with common terms will be co-located [21]. However, given the size and distribution of the data (many billions of triples, with terms occurring according to a power-law [82]) the rendezvous peers will suffer from highly unbalanced load distributions.

Note that standard techniques for load-balancing [60] will not work in our situation, since: (a) some popular URIs appear in very many triples, thus we have more items sharing one key than can fit in a single node so replication and caching will not help, and (b) we need all items with the same key to meet each other, so sub-dividing the keyspace over multiple responsible nodes will also not help.

[41] have an iterative forward-chaining procedure similar to ours but do not address load-balancing issues. [59] propose a backward-chaining algorithm which seems promising, but no conclusions can be drawn given the small dataset ($10^4$ triples) and atypical evaluation queries. [14] perform limited reasoning over the locally stored triples and introduce a policy to deal with load-balancing issues, but only compute a fraction of the complete closure.

### 8.2.2 Federated reasoning

[102] introduce networked graphs, allowing transparent data integration and querying of remote RDF endpoints, with an initial evaluation over small datasets. DARQ [91] uses a similar approach but adds query optimisation based on endpoint descriptions and statistics, which improves query performance. Neither approach addresses inferencing.

[108] perform distributed description logics reasoning; the system relies on manually created ontology mappings, which is quite a limiting assumption, and its performance is not evaluated. [103] distribute the reasoning rules instead of the data: each node is only responsible for performing a specific part of the reasoning process. Although efficient by preventing duplicate work, the weakest node in this setup becomes an immediate bottleneck and a single-point-of-failure, since all data has to pass all nodes for the system to function properly.

[54] use a modified ruleset which allows reasoning using a single pass over the data. This approach is orthogonal to ours and could be integrated into MARVIN, since we treat each single reasoner as a black box.
8.3 Our approach: Divide-conquer-swap

MARVIN operates using the infinite main loop shown in Algorithm 7, which is run on a set of compute nodes. These nodes have the same functionality and we will also refer to them as peers. In this loop, nodes grab some partition of the data, compute the closure on their partition, and then re-partition and swap with another node to find more inferences. The nodes keep re-partitioning, solving, and swapping to find all solutions. This “divide-conquer-swap” approach raises two questions:

- Does the approach converge and do we ever reach logical completeness?
- Is the approach efficient and scalable: what is the base performance of this model and how much performance gain is given by additional compute resources?

We will answer these questions in the next two sections. First, in Section 8.4, we show that the model does indeed reach completeness eventually. Next, in Section 8.5 we show that the distributed computation generates many duplicate triples which need to be detected and removed for efficient performance. Finally, in Section 8.6 we show that this model is inefficient if nodes exchange data randomly and we show how to strongly improve efficiency without sacrificing load-balance through our SpeedDate technique.

Algorithm 7 Divide-conquer-swap

1. The input data is divided into smaller chunks, which are stored on a shared location.
2. A large number of reasoners is started on several computational nodes.
3. Each node reads some input chunks and computes the corresponding output of this input data at its own speed.
4. On completion, each node selects some parts of the computed data and the input data, and sends it to some other node(s) for further processing. Asynchronous queues are used to avoid blocking communication.
5. Each node copies (parts of) the computed data to some external storage where the data can be queried on behalf of end-users. These results grow gradually over time, producing anytime behaviour.
8.4 Eventual completeness

In this section we will provide a qualitative model to study the completeness of MARVIN. Assuming a sound external procedure in the “conquer” step, overall soundness is evident through inspection of the basic loop, and we will not discuss it further.

The interesting question is not only whether MARVIN is complete: we want to know to which extent it is complete, and how this completeness evolves over time. For such questions, tools from logic do not suffice since they treat completeness as a binary property, do not analyse the degree of completeness and do not provide any progressive notion of the inference process. Instead, an elementary statistical approach yields more insight.

Let $C^*$ denote the deductive closure of the input data: all triples that can be derived from the input data. Given MARVIN’s soundness, we can consider each inference as a “draw” from this closure $C^*$. Since MARVIN derives its conclusions gradually over time, we can regard MARVIN as performing a series of repeated draws from $C^*$ over time. The repeated draws from $C^*$ may yield triples that have been drawn before; peers could re-derive duplicate conclusions that had been previously derived by others. Still, by drawing at each timepoint $t$ a subset $C(t)$ from $C^*$, we gradually obtain more and more elements from $C^*$.

In this light, our completeness question can be rephrased as follows: how does the union of all sets $C(t)$ grow with $t$? Will $\bigcup_t C(t) = C^*$ for some value of $t$?

At which rate will this convergence happen? Elementary statistics tells us that if we draw $t$ times a set of $k$ elements from a set of size $N$, the number of distinct drawn elements is expected to be $N \times (1 - (1 - k/N)^t)$. Of course, this is the expected number of distinct drawn elements after $t$ iterations, since the number of drawn duplicates is governed by chance, but the “most likely” (expected) number of distinct elements after $t$ iterations is $N \times (1 - (1 - k/N)^t)$, and in fact the variance of this expectation is very low when $k$ is small compared to $N$.

In our case, $N = |C^*|$, the size of the full closure, and $k = |C(t)|$, the number of triples jointly derived by all nodes at time $t$, so that the expected completeness $\gamma(t)$ after $t$ iterations is:

$$\gamma(t) = (1 - (1 - \frac{|C(t)|}{|C^*|})^t)$$

Notice that the boundary conditions on $\gamma(t)$ are reasonable: at $t = 0$, when no inference has been done, we have maximal incompleteness ($\gamma(0) = 0$); for trivial problems where the peers can compute the full closure in a single step (i.e. $|C(1)| = |C^*|$), we have immediate full completeness ($\gamma(1) = 1$); and in general if the peers are more efficient (i.e they compute a larger slice of the closure at each iteration), then $|C(t)|/|C^*|$ is closer to 1, and $\gamma(t)$ converges faster to 1, as expected. The graph of unique triple produced over time, as predicted by this model, is shown in figure 8.1. The predicted completeness rate fits the curves that we find in experimental settings, shown in the next section.
This completeness result is quite robust. In many realistic situations, at each timepoint the joint nodes will only compute a small fraction of the full closure \((C(t) \ll |C^*|)\), so that \(\gamma(t)\) is a reliable expectation with only small variance. Furthermore, completeness still holds when \(|C(t)|\) decreases over \(t\), which would correspond to the peers becoming less efficient over time, through for example network congestion or increased redundancy between repeated computations.

Our analytical evaluation shows that reasoning in MARVIN converges and reaches completeness eventually. Still, convergence time depends on system parameters such as the size of internal buffers, the routing policy, and the exit policy. In the next section, we report on empirical evaluations to understand the influence of these parameters.
8.5 Duplicate detection and removal

Since our aim is to minimise the time spent for deduction of the closure, we should spend most time computing new facts instead of re-computing known facts. Duplicate triples can be generated for several reasons like redundancy in the initial dataset, sending identical triples to several peers or deriving the same conclusions from different premises.

In reasonable quantities duplicate triples may be useful: they may participate, in parallel, in different deductions. In excess, however, they pose a major overhead: they cost time to produce and process and they occupy memory and bandwidth. Therefore, we typically want to limit the amount of duplicate triples in the system.

To remove duplicates from the system, they need to be detected. However, given the size of the data, peers cannot keep a list of all previously seen triples in memory: even using an optimal data structure such as a Bloom filter [18] with only 99% confidence, storing the existence of 8 billion triples would occupy some 9.5GB of memory on each peer.

We tackle this issue by distributing the duplicate detection effort, implementing a one-exit door policy: we assign the responsibility to detect each triple’s uniqueness to a single peer, using a uniform hash function: \( exit\_door(t) = \text{hash}(t) \mod N \), where \( t \) is a triple and \( N \) is the number of nodes. The exit door uses a bloomfilter to detect previously encountered triples: it marks the first copy of each triple as master copy, and removes all other subsequent copies.

For large numbers of nodes however, the one-exit door policy becomes less efficient since the probability of a triple to randomly appear at its exit door is \( \frac{1}{N} \) for \( N \) number of nodes. Therefore, we have an additional and configurable sub-exit door policy, where some \( k \) peers are responsible for explicitly routing some triples to an exit door, instead of waiting until the triples arrive at the designated exit door randomly.

A final optimisation that we call the dynamic sub-exit door policy makes \( k \) dependent on the number of triples in each local output buffer - raising \( k \) when the system is loaded and lowering it when the system is underutilized. This mechanism effectively works as a pressure valve, relieving the system when pressure gets too high. This policy is implemented with two thresholds: for any node, if the number of triples waiting to be shipped out exceeds \( t_{\text{upper}} \) then we set \( k = N \), if it is below \( t_{\text{lower}} \) then we set \( k = 0 \).

8.6 Efficient deductions

In this section we address the problem of finding inferences efficiently in a distributed system. As explained, to draw some conclusion, all triples involved in the deduction need to be co-located on the same machines.

We showed in Section 8.4 that randomly exchanging triples ensures that each
8.7. EXPERIMENTAL RESULTS

combination of triples is co-located in a machine at some point in time. However, random exchanges are inefficient since also irrelevant triples are co-located at one machine. With increasing number of nodes the chance for relevant triples to be co-located decreases strongly.

Assigning deterministic rendez-vous points to each triple (sending it to some specific peer) as is done in DHT-based approaches is more efficient, but suffers from load-balancing problems, as mentioned in Section 8.2. We use the SpeedDate algorithm, which was described in chapter 4.

8.7 Experimental results

The SpeedDate algorithm is focused on maximising the number of triples that meet with their “buddies”, to allow nodes to produce inferences. The overall Marvin system implements the SpeedDate routing strategy and performs the actual reasoning phase through an arbitrary off-the-shelve RDF/OWL reasoning library. As explained before, the Marvin system also includes the one-exit door policy for duplicate detection and removal.

We have implemented Marvin in Java, on top of Ibis, a high-performance communication middleware [130]. Ibis offers an integrated solution that transparently deals with many complexities in distributed programming such as network connectivity, hardware heterogeneity, and application deployment.

Experiments were run on the Distributed ASCI Supercomputer 3 (DAS-3), a five-cluster grid system, consisting in total of 271 machines with 791 cores at 2.4Ghz, with 4Gb of RAM per machine. All experiments used the Sesame in-memory store with a forward-chaining RDFS reasoner. All experiments we limited to a max. runtime of one hour, and were run on smaller parts of the DAS-3, as detailed in each experiment.

The datasets used were RDF Wordnet\(^2\) and SwetoDBLP\(^3\). Wordnet contains around 1.9M triples, with 41 distinct predicates and 22 distinct classes; the DBLP dataset contains around 14.9M triples, with 145 distinct predicates and 11 distinct classes. Although the schemas used are quite small, we did not exploit this fact in our algorithm (eg. by distributing the schemas to all nodes a priori) because such optimisation would not be possible for larger or initially unknown schemas.

8.7.1 Baseline: null reasoner

To validate the behavior of the baseline system components such as buffers and routing algorithms, we created a “null reasoner” which simple outputs all its input data. We thus measure the throughput of the communication substrate and the overhead of the platform.

\(^{2}\)http://larkc.eu/marvin/experiments/wordnet.nt.gz
\(^{3}\)http://larkc.eu/marvin/experiments/swetodblp.nt.gz
In this setup, the system reached a sustained throughput of 72.9 Ktps (thousand triples per second), with a sustained transfer rate of 26.5 MB/s per node. Typically, just indexing RDF data is slower (some 20–40 Ktps) [17], and reasoning is even more computationally expensive. Therefore, we can expect the inter-node communication (in the network used during our experiments) not to be a performance bottleneck.

8.7.2 Scalability

We have designed the system in order to scale to a large number of nodes. The Ibis middleware is based on solid grid technology which allows MARVIN to scale to a large number of nodes. Figure 8.2 shows the speedup gained by additional computational resources (using random routing, on the SwetoDBLP dataset), showing the number of unique triples produced for a system of 1–64 nodes.

Figure 8.2 shows the growth curves for different numbers of nodes. The sharp bends in the growth curves (especially with a small number of nodes) are attributed to the dynamic exit doors opening up: having reached the $t_{upper}$ threshold, the nodes start sending their triples to the exit door, where they are counted and copied to the storage bin.

<table>
<thead>
<tr>
<th>nodes</th>
<th>time (min)</th>
<th>speedup</th>
<th>scaled speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1.47</td>
<td>0.73</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>1.69</td>
<td>0.42</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>2.20</td>
<td>0.28</td>
</tr>
<tr>
<td>16</td>
<td>9.5</td>
<td>4.63</td>
<td>0.29</td>
</tr>
<tr>
<td>32</td>
<td>6.2</td>
<td>7.10</td>
<td>0.22</td>
</tr>
<tr>
<td>64</td>
<td>3.4</td>
<td>12.94</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 8.1: Speedup for SwetoDBLP dataset
Table 8.1 shows the time needed to produce some fixed number of triples in the SwetoDBLP dataset (namely, 20M triples). The same result is shown graphically in Figure 8.2. The table and graph show the amount of time needed over different numbers of nodes, the corresponding speedup (total time spent compared to time spent on single node) and the scaled speedup (speedup divided by number of nodes). A perfect linear speedup would equal the number of nodes and result in a scaled speedup (speedup divided by number of nodes) of 1. To the best of our knowledge no relevant literature is available in the field to compare these results, but a sublinear speedup is to be expected in general. As we can see, the system scales gracefully.

8.7.3 Duplicate detection and removal

We have experimented with three different settings of the dynamic sub-exit door: “low” where \( t_{\text{lower}} = \alpha, t_{\text{upper}} = 2\alpha \); “medium” where \( t_{\text{lower}} = 2\alpha, t_{\text{upper}} = 4\alpha \); “high” where \( t_{\text{lower}} = 4\alpha, t_{\text{upper}} = 8\alpha \), where \( \alpha \) is the number of input triples / \( N \).

These different settings were tested on the Wordnet dataset, using 16 nodes with the random routing policy. The results are shown in figure 8.3. As we can see, in the “low” setting, the system benefits from having low tolerance to duplicates: they are removed immediately, leaving bandwidth and computational resources to produce useful unique new triples. On the other hand, the duplicate detection comes at the cost of additional communication needed to send triples to the exit doors (not shown in the figure).

8.8 Conclusion

We have presented a platform for analysing Web data, with a focus on the Semantic Web. To process and interpret these datasets, we need an infrastructure that can scale to Web size and exploit the available semantics. In this chapter, we have focused on one particular problem: computing the deductive closure of a dataset through logical reasoning.

We have introduced Marvin, a platform for massive distributed RDF inference. Marvin uses a peer-to-peer architecture to achieve massive scalability by adding computational resources through our novel divide-conquer-swap approach. Marvin guarantees eventual completeness of the inference process and produces its results gradually (anytime behaviour). Through its modular design and its built-in instrumentation, Marvin provides a versatile experimentation platform with many configurations.

We have experimented with various reasoning strategies using Marvin. The experiments presented show that Marvin scales gracefully with the number of nodes, that the communication overhead is not the bottleneck during computation, and that duplicate detection and removal is crucial for performance.
Figure 8.3: Triples derived using the dynamic exit-door policy
Part III

Applications
In the final part of this thesis, we will present a scalable and distributed knowledge sharing application. Some of the methods previously presented were used for the implementation of the fully functional distributed discovery system of this application.
Chapter 9

The OpenKnowledge system

The techniques developed in chapters 2, 3 and 5 were applied in the design of the discovery mechanisms of the OpenKnowledge system.

With regard to the goals set in the introduction, this work refers to scalability, interoperability, openness and control.

This chapter consists of work also published in the following papers:

- “The OpenKnowledge Kernel”, presented in CESSE ’07 [3]
- “The OpenKnowledge System: an interaction-centered approach to knowledge sharing”, presented in COOPIIS ’07 [112]
- “Open Knowledge - Coordinating Knowledge Sharing through Peer-to-Peer Interaction”, presented as an invited paper in LADS 2007 [100]
- “Models of Interaction as a Grounding for Peer-to-Peer Knowledge Sharing”, published in LNCS Advances in Web Semantics [99]
abstract

The information that is made available through the semantic web will be accessed through complex programs (web-services, sensors etc.) that may interact in sophisticated ways. Composition guided simply by the specifications of programs’ inputs and outputs is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs. These interaction models, however, are traditionally viewed as a consequence of service composition rather than as the focal point for facilitating composition. We describe an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources, discovery and interaction control.

9.1 Introduction

The pool of potentially available knowledge on the Internet is immeasurably large. It is fed by the traditional Web: by application programs feeding data onto the Web, by Web services accessed through various forms of application interface, by devices that sense the physical environment, and so on. It is consumed in a wide variety of ways and by diverse mechanisms (and of course consumers may also be suppliers). The aspiration of OpenKnowledge is to allow knowledge to be shared freely and reliably, regardless of the source or consumer. Reliability here is interpreted as a semantic issue. The Internet is in the fortunate situation that physical and syntactic reliability have been solved to satisfactory degrees, making semantic reliability the main challenge. Semantic reliability means that we want the meaning ascribed to knowledge that is fed into the pool, to be preserved adequately for the purposes of consumers.

Of course such “open knowledge sharing” is an aspiration that we know to be unattainable, in the strong sense where all knowledge supplied can be consumed with perfect freedom and reliability. Globally consistent common knowledge is impossible to guarantee in an asynchronous distributed system\(^1\). Extensive standardization on ontologies is limited by the level of detail at which we can gain a meaningful consensus on the use of domain-specific terminology. Furthermore, the volume of knowledge transfer is limited by bandwidth, while local storage is limited by hardware capacity.

\(^1\)even if it were a philosophically and culturally coherent notion.
Interaction-specific knowledge sharing: The good news is that only a small proportion of the pool of available knowledge will be of use to any given consumer, since each must have an upper limit on how much knowledge it can process. A pragmatic aim of open knowledge sharing, then, is to obtain knowledge appropriate to the activities in which each consumer wants to engage, while maintaining free and (adequately) reliable connections between suppliers and consumers.

The standard way in which activities (and their sequencing) are described is via process languages like BPEL[9] or LCC[97], since no complex activity can be represented formally without modeling its temporal structure. In principle, we could use (models of) these activities to limit the scope of knowledge that we attempt to share. There is a problem however: activity models are themselves knowledge that must be shared. In other words, when an item of knowledge is openly shared in the context of some common activity it is necessary for the supplier and consumer to have knowledge of that context, otherwise there is no benefit (in terms of reliable knowledge sharing) from the activity focus.

For this reason the OpenKnowledge project has at its core a mechanism for sharing models of activities that require interaction across the Internet. We refer to such models as interaction models[97]. We expect that communities of practice will naturally form around collections of interaction models and that these communities can be stabilized by a mechanism for their rapid sharing across peer groups. Notice that this is explicitly an interaction-centered approach to knowledge sharing, as opposed to the traditional data-centered approach. Unlike knowledge in general, which is unbounded even for communities of practice, we assume that the key activities of many communities can be described by a bounded number of interaction models. Furthermore, since the purpose of an interaction model is to perform a useful activity, we assume that through widespread use of interaction models across a community it will be possible to assess (both at an individual level and statistically across a community) how fit each model is for a given purpose. Again, such fitness-for-purpose metrics have proved to be very difficult for the traditional data-centric views.

By building a system, we have demonstrated that sharing interaction models at very low cost to consumers and suppliers is possible. The novelty of this system is that each interchange of knowledge is made in the context of the (shared) interaction model. We then address the (unavoidable) tasks of ontology mapping, query routing, etc. using algorithms that are comparatively simple because they can (at no additional cost) use knowledge about the structure of the interaction and the ways in which it has been performed (successfully or unsuccessfully) within a peer group. Notice that this is the inverse of many traditional approaches to knowledge sharing in which one standardizes the semantics of the shared knowledge independent of the activity context across peers.

The system is completely distributed using P2P technology. Each peer that participates in the OK system will at least run a piece of code that we call the OpenKnowledge Kernel[3] enabling the base functionality to find these interactions.
and the code or peers that enable to run the services. More precisely, the system is focused on efficiently sharing and finding these formally described interaction models (IMs) together with pointers to either the code for the services or peers that can execute the services. We call these services OK components (OKCs). The IMs together with the OKCs are efficiently stored and retrieved in a P2P network. Besides this, due to the fact that the tasks are formally described, the OK system offers the functionality to coordinate a task by controlling the process flow between OKCs, (i.e., by executing the IM, selecting OKCs to fulfill a role, finding alternative OKCs in case of failure, and making sure the IM is followed by all OKCs). The users can publish IMs, write interfaces to services, and subscribe these interfaces to play roles in the IMs. The system helps these users by providing tools to ease re-use of existing IMs or by helping connect two services via mappings in case the output of one does not match the input of the other.

The remainder of this chapter is as follows: first, in section 9.2 we give a schematic overview of the relevant literature on distributed knowledge systems. After that, section 9.3 describes the functionality of the system by following an extensive example. In section 9.4, the architecture of the OpenKnowledge Kernel and the services are shown that make up the OpenKnowledge system. We summarize our work in section 9.5.

9.2 Relevant literature

Clearly, many others have previously identified the goals of reliably sharing knowledge freely and reliably, regardless of the source or consumer. In this chapter, we will not discuss the plethora of work in the dominant data-oriented attempts at solving this problem, such as data-integration[69], schema and ontology mapping[92, 110], data-mediators[46], etc. Instead, in this section we discuss some of the approaches that have also taken an interaction-oriented approach: web-services, grid-services and multi-agent systems. Although typically data-centric, we also include P2P systems in our comparison, because the OpenKnowledge architecture has strong P2P characteristics.

We do not aim to provide a full-scale literature study here. Instead, we identify the key ideas behind each of these approaches, and argue why OpenKnowledge occupies a unique niche in this landscape.

Web Services. Perhaps the most closely related effort to OpenKnowledge is the work on web-services[28, 81]. The aim of web-services is to enable invoking and executing of services in a distributed, scalable and interoperable manner. The work on semantic web-services[121, 86] adds to this the goals to automatically locate and compose such services in an open and heterogeneous environment like the Web.

Both approaches (web-services and OpenKnowledge) use the principle that if the services are formulated into information objects (web-service descriptions either
purely syntactic, such as WSDL[27] or semantic such as WSDL-S[114], OWL-S[76] or WSMO[42]), then they can also be the subject of reasoning tasks for search and composition.

The OpenKnowledge approach is in some ways more flexible than the web-services approach, but in other ways more restricted. Semantic web-service work aims at automatic on-line composition of simple services into complex services, by means of intelligent algorithms (e.g. based on configuration[124] or planning[126, 23]), whereas, OpenKnowledge restricts itself to executing predefined “work-flows” of services (the “interaction models” to be discussed later in this chapter). The only decision that OpenKnowledge makes at run-time is which instance of a service is executed; that is, which agent providing the service will be used (i.e. “recruiting”, not composition).

This recruiting aspect of OpenKnowledge is more general than the web-service architecture because it separates the advertising of a service from the execution of a service. In the web-service architecture, it is generally assumed that advertisements of service functionality are accompanied with the name of the executor of the service. In short: the matching goals of both approaches are the same (finding a service that matches a given functionality), while the composition goals of both approaches are different: OpenKnowledge aims to recruit peers to execute predefined work-flows, whereas semantic web-services aims to automatically compose complex work-flows out of atomic services.

Furthermore, OpenKnowledge explicitly acknowledges the need for approximate matching of service requests with advertisements, whereas this is only marginally the case in the semantic web-service world[4], and entirely absent in regular web-services.

Finally, OpenKnowledge aims explicitly for a distributed storage model for the work-flows and service descriptions, whereas all the dominant web-service architectures (UDDI[85] for regular web-services, WSMX[51] for semantic web-services) assume a centralised architecture.

Grid-Services. The general area of grid-services is even less well circumscribed than web-services, hence it is more difficult to make a crisp comparison. Literature on Grids[45] often align their approaches to the service-oriented architecture (SOA)[40]. In contrast to web-services, grid-services are typically organized in fixed work-flows. This makes them more similar to the OpenKnowledge approach, however, grid-services emphasise various aspects that are ignored in OpenKnowledge: long-term stability of services, provenance, quality of service and resource monitoring. Similar to web-services, grid-services differ from Open Knowledge by advertising a service functionality together with the identification of the service-provider; OpenKnowledge decouples these two and hence allows for a separate “recruiting” step. Finally, and perhaps most importantly, most grid-systems provide only a centralized mechanism for advertising services and work-flows, while OpenKnowledge aims for a fully distributed mechanism.
In particular, the myGrid project [117] is in many respects close to the goals of OpenKnowledge in its use of pre-configured work-flows and its approach to manual composition of such work-flows. However, it relies on centralized storage of such work-flow patterns, which is in sharp contrast with the fully distributed architecture of OpenKnowledge.

Peer-to-peer systems. Obviously, OpenKnowledge is close in spirit to the work on peer-to-peer (P2P) systems. The central P2P ideas of distributed storage, lack of centralized address registers and the symmetric roles of every peer as both provider and requester, are fully adopted by OpenKnowledge. Nevertheless, OpenKnowledge makes two important deviations from most P2P systems. First, most P2P systems aim at data sharing, whereas OpenKnowledge aims at service sharing. Of course, data sharing is simply a special case of service sharing (namely sharing a data-access service), making the OpenKnowledge system more generic. Secondly, OpenKnowledge is in the small, but rapidly growing, family of semantic P2P systems[123], which use rich descriptions of the content that each peer has to offer for purposes of routing queries through the network.

Agents A final class of closely related systems is that of multi-agent systems. In general, there is a superficial similarity between multi-agent and P2P systems: distributed sets of autonomous processes exchanging information. However, on closer inspection, there are rather significant differences. In particular, agent systems have highly structured architectures inside each agent often relying on cognitive metaphors for their architectural constructs (such as the Believes, Desires and Intentions (BDI) architecture[93]). P2P systems typically treat their peers as atomic. Finally, agent-systems emphasize their pro-active nature (autonomously reacting on their changing environment), while P2P systems, including OpenKnowledge, assume more classical reactive stance.

The differences and similarities described above are all summarized in Table 9.1. This table shows that OpenKnowledge inherits many aspects from other approaches but also occupies a particular niche, having features not fully explored by others.

Both grid-services and P2P may also be interpreted not as systems that provide user-functionality (as we did above and in table ), but as platforms on which other systems that offer user-functionality can be developed. This view enables a comparison of these two systems with the Web (which is also a platform for applications, and not an application in itself).

Figure 9.2 shows a comparison between the fundamental platforms that we have discussed so far (the Grid, P2P and the Web) and attempts to express the additional features that the OpenKnowledge system will beyond these existing platforms. The term cloud is clustered by platform and sometimes there is an overlap. The terms inside the circle of the figure are characteristics of the OpenKnowledge system, and
### Web-Services

**Similarities:**
- Service-oriented,
- Distributed,
- Automated search
- Based on semantic descriptions

**Differences:**
- **Web-Services**
  - Composition of atomic services
  - Fixed link to executing party
  - Centralised advertising
  - Equivalence matching
- **OpenKnowledge**
  - Predefined workflows
  - Dynamic recruiting
  - Distributed
  - Approximate matching

### Grid-Services

**Similarities:**
- Service-oriented,
- Fixed workflows
- Distributed

**Differences:**
- **Grid-Services**
  - Provenance
  - QoS
  - Resource monitoring
  - Centralised advertising
  - Fixed link to executing party
- **OpenKnowledge**
  - Absent
  - Reputation mechanisms
  - Absent
  - Distributed
  - Dynamic recruiting

### Peer-to-Peer Systems

**Similarities:**
- Distributed,
- Scalable,
- Symmetric roles of each peer

**Differences:**
- **P2P Systems**
  - Aimed at data-sharing
  - Independent of content
- **OpenKnowledge**
  - Service sharing
  - Exploit semantics

### Multi-Agent Systems

**Similarities:**
- Distributed,
- Symmetric roles of each peer

**Differences:**
- **Multi-Agent Systems**
  - Cognitive architecture
  - Central brokers
  - Pro-active behaviour
- **OpenKnowledge**
  - None
  - Scalable discovery
  - Reactive

---

Figure 9.1: OpenKnowledge compared to other approaches
CHAPTER 9. THE OPENKNOWLEDGE SYSTEM

9.3 An extensive example describing the functionality of the OpenKnowledge system

From a user perspective, the OpenKnowledge system is a software bundle that allows a user to find, compose and execute tasks. Those tasks can be executed by users and/or software components. The tasks are described by Interaction Models (IM),
where each IM is a formally described set of roles together with the process-flow between those roles. Users subscribe their peer to play roles within an interaction. For example, the task of buying an item requires at least the seller and buyer roles, and perhaps a payment service role. We call instances of these roles (e.g. a particular seller or a particular buyer) **OK-Components (OKCs)**. If the roles are constrained by some external functionality, then services provide that functionality.

In other words, the OpenKnowledge system facilitates the process of bringing together providers and consumers by having an infrastructure where programmers can write tasks in the form of Interaction Models and implementations of roles in OKCs. The system takes care that these IMs and OKCs are robustly and efficiently stored and retrieved in a scalable way. Besides this, the system also helps the programmer to write new IMs. For example, when roles are combined into a new IM, the system can provide historical ontological mapping information when connecting these roles.

Now we will explain the functionality of the first OpenKnowledge system by going through an example where we show how a dictionary service can be created and used. The scenario that we describe in this section is made up of three peers, but many more could be added without changing the general idea. Each of the peers will also be part of the **Discovery and Team formation Service (DTS)** and subscribe to act as **coordinator**, both of which will be described later in the chapter. It is important to remember that many peers can play the role of a **coordinator** and that the **DTS** has a completely decentralized implementation, which does not introduce a performance bottleneck in the OpenKnowledge network. To simplify matters for this example we assume that these peers are always online, and the OK Kernel (the piece of software providing the basic functionality for the OpenKnowledge System) is being executed in each one of them.

### 9.3.1 Writing and publishing an IM

In figure 9.3 user A uses the OpenKnowledge System to develop an IM for the dictionary service, by describing an interaction between two roles. One role is used to query the service, called the **inquirer**, and the role used to provide the answer is the **oracle**. In this example, the IM is written in the LCC language[97] which is currently the only language supported by kernels running on the OpenKnowledge peers. Current work in the project is to also have support to other languages like BPEL. The LCC model can be read as follows:

1. $r(\text{inquirer}, \text{initial})$. This line states that the 'inquirer' role is the one that starts the interaction.

2. $r(\text{oracle}, \text{necessary}, 1)$. Statement indicating that at least 1 peer needs to play the oracle role.
3. \texttt{a(inquirer, ID2)::}. A statement giving the 'inquirer' role an identifier 'ID2' and the '::' means that the definition of the role starts after it.

4. \texttt{ask(W) => a(oracle, ID) <- toknow(W)}. If the user wants to know a definition for a word 'W' it can start the interaction by fulfilling the constraint \texttt{toknow(W)}. In LCC the '<-' symbol is used to indicate that after it a constraint is defined. When the constraint is satisfied (i.e. the user provided 'W'), a message 'textttask(W)' is sent to the 'oracle' role identified by 'ID' (note that \texttt{a(oracle, ID)} relates the role to an identifier). In LCC the '=>' symbol is used to indicate that a message (in this case \texttt{ask(W)}) is sent from the current role to another role (in this case the 'oracle').
9.3. AN EXTENSIVE EXAMPLE DESCRIBING THE FUNCTIONALITY OF THE OPENKNOWLEDGE SYSTEM

5. definition(\(W,D\)) \(<=\) a(oracle,ID). In this line the ‘inquirer’ waits for the oracle role (a(oracle,ID)) to send a message with the definition as content (definition(\(W,D\))). In LCC the ‘\(<=\)’ symbol is used to indicate that a message (in this case definition(\(W,D\))) should be expected from another role (in this case the ‘oracle’ role).

6. null \(<=\) show(\(W,D\)). When the ‘oracle’ sent the message to this role, this statement shows the answer to the user. In this case show is a special constraint which is understood by the system to show a message (in this case with the query: \(W\) and the answer: \(D\)) in the user interface. null means that nothing happens after the constraint show(\(W,D\)) is fulfilled.

7. a(oracle,ID)::. Gives the ‘oracle’ role identifier ‘ID’ and starts to give its definition.

8. ask(\(W\)) \(<=\) a(inquirer,ID2). This line makes the ‘oracle’ role wait for a message ask(\(W\)) from the ‘inquirer’.

9. definition(\(W,D\)) \(=>\) a(inquirer,ID2) \(<-\) define(\(W,D\)). When the ‘oracle’ got the ‘ask’ message (previous line is executed), it will try to fulfill the ‘define(\(W,D\))’ constraint, and if that is true, a message with the content definition(\(W,D\)) is sent to the ‘inquirer’.

Now that a user A wrote down the IM, they should provide some keywords to describe the functionality of the IM. These keywords are needed by the DTS to index them in order to be retrieved by other peers. In this case, A decides to give the keywords ‘oracle, wordnet, dictionary, words’. The keyword ‘wordnet’ already indicates how the user will implement the ‘oracle’ OKC, namely by having a Wordnet dictionary. Our current work tries to extend the ways to describe the functionality of an IM, for example by providing concepts from ontologies instead of keywords. Now that the IM is ready and the keywords are provided, the user can decide to publish it on the OpenKnowledge network by connecting to the network and pressing the ‘Publish Interaction Model’ button. The DTS will make sure it is scalably stored and indexed by the provided keywords.

9.3.2 Creating and publishing OKC’s

Besides writing the IM in the previous section, user A also writes the OKCs that implement both roles in the IM respectively. Currently, the user A has to implement their OKC by writing some code to a specific Java API. In simple terms, the methods in the Java source code should match the names and the arguments of the constraints in the roles, which are toknow(\(W\)) and show(\(W,D\)) for the ‘inquirer’ role and define(\(W,D\)) for the ‘oracle’ role. Note that here we assume \(W\) and \(D\) are of type STRING, where in the extended LCC language also types are supported, meaning that the definitions would be something like show(\(W:\text{STRING},D:\text{STRING}\)). After user
CHAPTER 9. THE OPENKNOWLEDGE SYSTEM

Figure 9.4: User interface for OKC creation. The user loads an IM and attaches the java implementation of the role constraints. The interface shows which constraints for the roles are implemented (in this case for the ‘inquirer’ role both the constraints show() and toknow() are implemented.

A has implemented the interfaces, (s)he opens the window from the OpenKnowledge Kernel software as shown in figure 9.4.

The user loads their IM and attaches the java implementations of the role constraints. The interface shows which constraints for the roles are implemented. In the figure, for the ‘inquirer’ role both the constraints show() and toknow() are implemented. Also the OKCs may be described by a set of keywords, because they can be used as role implementations for other IMs and therefore need to be indexed so that they can be retrieved by the DTS. The intuition behind this is that an OKC implementing a credit-card payment service can be used in many IMs. Also these
9.3. AN EXTENSIVE EXAMPLE DESCRIBING THE FUNCTIONALITY OF
THE OPENKNOWLEDGE SYSTEM

keywords can be used in the OKC selection process that allows a user to select their preferred OKCs after multiple matches have been found to an IM. For example, it can be that two OKCs exactly match the same ‘oracle’ role but one delivers results in English and the other in Spanish.

By clicking the ‘Create OpenKnowledge Component’ button, the OKC is created and ready to be used. By sending a 'subscribe' message to the DTS (not shown in the figures), it tells the network that it is able to execute the role of ‘oracle’ for the given IM. Given that the user used Wordnet as the underlying implementation, it annotates the OKC with the keywords ‘dictionary, english, wordnet, lookup’ (not shown in the figures). Besides this, A decides to publish the ‘inquirer’ OKC to the network, so that other users also may download it and run it on their own machines.

9.3.3 Searching for IMs and OKCs

Peer B wants to find a service that will allow it to find definitions of words in Spanish. It opens the window from the OpenKnowledge Kernel as shown in figure 9.5.

On the lefthand side of the figure, user B’s directory is shown containing i) the OKCs user B has stored locally, ii) an overview of the OKCs it has been subscribed to and iii) an overview of its favorite searches. On the top, the query field is shown to find IMs. In this case, in the beginning (s)he searches for IMs matching to the word ‘oracle’. The middle shows the found IMs together with their roles. Here B is curious and unfolds the IM described by the keywords ‘expert, oracle, search’ and sees that it does not fulfill the intentions. Therefore, B reformulates the search (not shown in the figures) by the query: ‘dictionary, spanish’. Imagine that user B finds the IM and the ‘inquirer’ OKC written by user A and decides to download it and tells the DTS that it is willing to play the role.

9.3.4 Team formation and execution

Given that in the previous steps A and B have both told the DTS that by subscribing their OKCs that they are willing to play the roles of ‘oracle’ and ‘inquirer’ respectively, the DTS knows that all roles are instantiated meaning that there are enough peers to start the interaction. Now imagine that another user C also published an OKC that is able to fulfill the role of ‘oracle’, but has annotated its OKC with the keywords ‘dictionary, spanish’. So now there are three peers ready to play. The DTS selects a coordinator peer from the pool of peers. This is currently selected randomly (but current ongoing work is to make it reputation-based). This coordinator receives a message from the DTS with the three peers, their OKC descriptors and the IM. The coordinator now can start the team formation process.

The coordinator sends each peer the list of peers willing to play together with their OKC descriptions. Now the peers can select, automatically or with the user in the loop (depends on the OKC implementation), with whom to play. Assume that both the Spanish and English oracles have automatic selection process saying
Figure 9.5: User interface showing the IM and OKC search panels. On the left-hand side, a directory is shown containing (1) the OKCs it has stored locally, (2) an overview of the OKCs it has been subscribed to and (3) an overview of its favorite searches. On the top, the query field is shown to find IMs. The middle shows the found IMs together with their roles. Here a user unfolded the IM described by the keywords ‘expert, oracle, search’. By double-clicking on a role, a search is started for OKCs that implement the roles. The user can also search for OKCs in the right panel. The bottom of the image shows a schematic overview of the role dependencies in the selected IM.
that they always like to play with whomever. However, the inquirer has user B in
the loop, where the user selects the peer from user C, because its OKC description
matches its wishes and sends its preferences back to the coordinating peer. Now that
the coordinator has (within a certain time-out) received enough replies to start the
interaction, it starts executing it. The coordinator sends a message to Peer B which
solves the constraint by asking the user (using a visualizer showing the constraint
to the user). The word is sent back to the coordinator which continues parsing the
IM and reaches a constraint that must be satisfied by the dictionary role to give the
word definition. The coordinator sends the constraint to Peer C which solved it and
returns the definition in a message. The coordinator continues parsing and finds a
constraint in which the querier role must show the user the word definition. It sends
Peer B a message with this constraint and it is solved by showing the query results
to the user. The IM is finished at this point, so the coordinator sends a message to
each peer so they can stop the OKC instances.

As said, this example demonstrates the functionality of the system, but it is very
simple. The interface presented is only one of the many possible interfaces, because
we have designed the architecture to be as independent as possible from the user
presentation system.

9.3.5 Other examples

Some interesting examples can be made within the trade domain, like an interaction
model for a transaction of goods. Somebody may publish an IM that contains the
process-flow between a seller, a buyer and a payment service. Peers can subscribe
to these roles and when all roles are instantiated the interaction starts. The Coordinator (see section 9.4) initiates the interaction and coordinates it. Es-
pecially in this case, all role-players may want to have a trustworthy controller, and
can specify the requirements for a coordinator when subscribing to an OKC.

Another example comes from a case study that we undertook in the bio-informatics
domain[132]. In that paper we present a system that can be used to analyse real data
of relevance to the structural bio-informatics community where comparative mod-
els of yeast protein structures from different resources are analysed for consistency
between them. The interaction model described in that paper, written in the LCC
language, describes the interaction between the roles of data collector, receiver and
source, that together perform the task. The difference with this approach compared
to the OK-system is that the implementation of the interaction model interpreter,
the Magenta system, is completely centralised. Also the interaction models and the
services are fixed and are therefore not dynamically discovered as in our system.
9.4 The architecture of the OpenKnowledge system

The OpenKnowledge system has a P2P architecture where each autonomous peer shares a common piece of software that we call the OpenKnowledge Kernel as already mentioned earlier. In this section we briefly introduce the kernel’s architecture. For a more detailed description we refer to [3]. Figures 9.6 and 9.7 show a schematic overview of the architecture and of a peer. We follow a bottom up approach, starting from the basic building blocks, namely OpenKnowledge Components (OKCs) and Interaction Models (IMs). An IM is a formal specification written in a language devised for this purpose (e.g., LCC[97] or BPEL2). IMs contain roles and define the interactions between them. Roles are implemented by OKCs. An OKC is comparable to a web service: it has an implementation and a standardized way to describe functionality. OKCs are mobile and are stored locally in OKC repositories, and in a distributed way via the DTS (see section 9.4.1). Once they are being executed as part of an IM, the OKC instances are stored locally in instance repositories. Each peer manages the OKCs it has stored locally and can also act as a coordinator of an interaction between OKCs. OKCs communicate with the coordinator via the communication layer. A user-interface is provided to access the basic OK functionality by the user; creating IMs and OKCs, search for IMs, download OKCs, subscribe OKCs to roles, team-formation and communication during interactions. The control manager provides execution control over the peer’s modules.

Besides these, there is a set of elements that are not essential for OpenKnowledge Kernel in every peer but are essential for the OpenKnowledge system as a whole. Therefore, a subset of the OpenKnowledge peers need to execute them. Some peers, acting as coordinators, need to interpret the IMs and coordinate the communication between OKCs. Furthermore, some services are provided for peers to aid in the interaction process. The Discovery and Team formation Service (DTS) stores IMs, OKCs, and their subscriptions. The Trust and Reputation Service (TRS) is used to gather information about other peers in order to guide the user in choosing interaction partners. The Mapping Service (MS) is used by peers when interacting with each other to aid in mutual understanding.

9.4.1 Services

The Kernel’s atomic functionality cannot be implemented using OKCs, so P2P services are used. The discovery and team formation service is an essential service, meaning that the system cannot operate without it. The DTS described previously is such an example which is currently implemented. Other services, namely the mapping service and the trust and reputation service are in the focus of our research, but are optional.

Discovery and Team formation Service (DTS)

Much of the functionality of the OK system relies on the DTS, its main responsibilities being the following:

- **IM Discovery** - the DTS is used to publish, discover and retrieve IMs.

- **OKC Discovery** - the DTS is also used to publish, discover and retrieve OKCs. This enables reusability thus providing scalable functionality. OKCs can be discovered either in the context of an already known IM or independently.

- **Role subscription** - peers can subscribe a locally stored OKC to play a role in an IM. Additional information such as annotations and restrictions concerning the other participants can be given along with the subscription.
• Coordinator subscription - peers may also subscribe to act as interaction co-ordinators.

• Team formation and interaction initialization - the DTS uses subscription information to form teams of OKCs, which will, potentially, participate in an interaction, and finds a subscribed coordinator to orchestrate them.

Mapping Service (MS)
An OKC needs to understand the other OKCs it is interacting with. Chances are that an OKC will not interact with all the other OKCs, therefore defining an a priori ontology seems unreasonable, given the complexity of the task. Furthermore, we want to achieve low entry cost, therefore, matching of one OKC’s terms to another must be done at runtime. The MS’s aim is to aid in this runtime process.

The MS is used in the searching and the interaction processes. When searching for IMs and OKCs it is used to map the text in their annotations to the query. When interacting, an OKC can use the MS to map those terms that another OKC is sending to its own terms. The MS taps into the information gathered from the system use, to provide community-supported mappings.

Trust and Reputation Service (TRS)
Since we are dealing with a completely decentralized system and flexible interactions, a service to maintain trust relationships greatly benefits the range of applications suitable for OpenKnowledge. To this end, a combination of personal preferences and community past experience needs to be taken into consideration when choosing whom to accept as interaction partner. Note, that although the policies governing this can be complex, our architecture is independent of the TRS implementation.

All that is needed from the user by the TRS is that it rates the interactions with other peers. This action is optional, but it is feasible to assume enough users will be willing to rate interactions seeing how other community-based rating systems such as FlickR and YouTube are being used. In return, users will be able to query the TRS in order to retrieve reputation data about other OKCs and peers with which (s)he might have to interact.

9.5 Summary
Much of the information that might be accessed in semantic webs is accessible through complex programs (web-services, sensors etc.) that may interact in sophisticated ways. Composition guided simply by specifications of programs’ input-output behaviours is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs.
These interaction models, however, are traditionally viewed as a consequence of service composition rather than as the focal point for facilitating composition. We have described an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources of interaction control.

We have described a system in which all knowledge exchange is supported by models of interaction shared between peers in an open environment. The task of constructing interaction models is similar in sophistication to programming, so repositories of models are built by programmers interested in supplying a new service to their peer community. Sharing of interaction models is achieved through query routing within peer groups, where potential suppliers advertise interactions and potential consumers query the network for services required. All peers have the capability to interpret interaction models so, once a peer identifies an interaction of interest, that interaction can be coordinated by any available peer in the system - thus distributing the computation. Knowledge sharing between peers is then conducted in the context of a specific interaction model. This allows us to simplify ontology mapping, since we are concerned only with mapping for a specific interaction context and peer group, to the point that limited forms of dynamic ontology mapping become possible. It also allows us to accumulate statistical information (shared between peers) about the frequency of co-occurrence of terms and peer identifiers in successful interactions. Some basic specifications for a variety of them appear in [96]. Facilities such as dynamic ontology mapping are, we believe, best viewed as extensions of the OpenKnowledge system (the focus of this chapter) with the choice of which extensions to use being made in the light of experience with the system. The direct contribution of this chapter is to present the first operational system of this kind. The secondary contribution of this chapter is to provide a new angle on service orchestration and ontology matching that re-interprets traditional methods for these tasks in a dynamic context.

Acknowledgements: This work has been supported by the FP6 OpenKnowledge project\(^3\).

\(^3\)http://www.openk.org/
Chapter 10

Conclusions

The fact that the Web now contains massive amounts of private and public data with well-defined semantics has led us to a set of goals for discovery systems: scalability, interoperability, openness, control and completeness.

We have developed a series of methods that fulfill these requirements:

First, a method to improve the performance of DHT-based indexes based on term popularity was presented (chapter 2). This method detects and does not index overly popular terms, trading completeness for scalability. Furthermore, it supports the goal of openness in participation, since it is based on a peer-to-peer network which any host can join.

Second, we have shown a method to create a scalable obfuscated distributed index based on a DHT using truncated secure hashes (chapter 3). Similar to the previous method, it contributes to the goal of openness in participation. Moreover, it provides the functionality of locating content without disclosing it, thus providing additional control.

Third, in chapter 4, a mechanism for scalable rendezvous was described. It combines a deterministic DHT-like mechanism to increase performance with a random exchange mechanism to balance load across nodes. In addition, it is eventually consistent, in the sense that at one point in time all items will have met all other items that they were meant to meet. Nevertheless, information owners have to trust this mechanism, since no node has particular ownership rights over any item. This affects control in a negative manner.

Fourth, an infrastructure for scalable Web Service discovery was presented in chapter 5. Publishing and querying is open. In general, Web Services aim at interoperability.

Fifth, a private sharing scenario and mechanism was described in chapter 6. The scalable distributed index previously described is combined with an anonymizing network to provide strong privacy guarantees. Non-understandable descriptors are published and negotiation happens in a peer-to-peer manner, allowing more control
as well as interoperability and openness since any access control policy that can be locally applied can be used.

Sixth, a peer-to-peer backward reasoning method was presented in chapter 7. Data is kept locally, which is of benefit to control, and descriptors are indexed by a DHT peer-to-peer network, which is of benefit to scalability. Similar to the previous approach, access policies are enforced locally, allowing interoperability and openness.

Seventh, in chapter 8, a forward reasoning system based on the rendezvous mechanism of chapter 4 was presented. It shares with it the same benefit regarding eventual completeness and drawback regarding control. The method allows various reasoners, related to openness and interoperability.

Eighth, the OpenKnowledge system (chapter 9) which aims at scalable knowledge sharing using an open set of peer-to-peer interactions was described. In this system, interoperability and control is provided through formalized models of interaction.

In table 10.1, a summary of the relation of these methods and systems to the goals specified in the introduction is given. We will use this table to revisit the research questions:

- **How can we design a scalable discovery mechanism for networked data?**
  We have shown the design of several discovery mechanisms for networked data. The first row of table 10.1 shows that all mechanisms and systems in this thesis are scalable.

- **How does the need for scalability relate to interoperability, openness, control and performance? Can a good compromise be reached?**
  In table 10.1, we can see no single method supporting all goals. As also indicated in the introduction, there seems to be a trade-off between these goals. We have developed discovery methods for various use-cases for which the compromise was good enough. Nevertheless, we cannot claim that one of the methods developed is suited for any purpose. Note that the fact that all methods fulfill the goal of scalability does not mean that it is easier to fulfill: it was a design decision for our research to make it a must-have.

Table 10.1: The chapters of this thesis positioned across the goals set in the introduction. A “+” means positive impact, a “-” means negative impact and a blank space means that the work is orthogonal to the goal.
Distribution did help develop scalable discovery mechanisms by spreading the computational load to all or a large subset of participants in the system. Notably, the biggest challenge encountered was data load balancing. Web data follows a very skewed distribution, thus distributing it among nodes is not straightforward. To make matters worse, precalculated knowledge about term popularity does not help very much in this context since most terms do not appear in dictionaries (for example “XML”). Even if that would be possible, using such precalculated knowledge would have an adverse effect on control, since the publisher of this knowledge should be trusted.

Our research has had an exploratory nature: we have developed some systems and have shown how they fare against the goals set. As such, table 10.1, provides a non-exhaustive view of the design space. Nevertheless, some issues are evident: The combination of scalability and completeness is hard since, typically, to find all answers, a large number of nodes has to be contacted. Methods have been developed to alleviate the problem, but they come at the cost of reduced control, since they require some sort of information dissemination either to locate content or organize nodes. This forms a triangular relationship scalability-control-completeness of difficult to reconcile goals. In section 10.2, we will give recommendations for further investigation.

10.1 Future work

In this section, we present possible extensions to the work presented in this thesis as well as new research questions that arise. Further investigation in the goals and in the relation with other fields will be discussed in section 10.2.

10.1.1 Privacy-enabled reasoning

The backward reasoning approach in chapter 7 can be combined with the rarity-based routing approach in chapter 2 and the privacy-preserving sharing in chapter 6.

The index for the backward reasoning can be enhanced by using the rarity-based technique since:

- The scalability of the index will improve by not indexing overly popular terms (i.e. “rdf:type”).

- The rarity-based index will provide information about the selectivity of each term in query time. In query resolution, it is better to start with the most selective terms. It is straightforward to enhance the rarity-based index with an additional primitive which would return the entries for the most rare term, given a set of terms as a query.
The privacy-preserving sharing method is directly applicable to the backwards reasoning architecture. The concept index can be obfuscated and the distributed query resolution/reasoning will include an additional step of matching access policies.

A mechanism to cope with overly popular ontologies also required. For example, we cannot expect that some peer will have the sole responsibility to handle all queries which involve a very popular ontology. On the other hand, delegating responsibility to other peers raises concerns about privacy, control and security. A solution would be a combination of caching and public key-based cryptography. The first would reduce the number of messages required to resolve queries while the second would allow a content distribution mechanism to disseminate signed popular (and public) content. How to handle content that is not public but yet popular remains an open question.

10.1.2 MaRVIN@home

The decentralized MaRVIN network can be used as a server backbone for a desktop grid. In this setting, the MaRVIN nodes partition and distribute the data and a large set of client nodes perform the actual computation (be that inferencing or other).

A desktop grid is well-suited for reasoning because:

- It is computationally easy to verify results. I.e. if an (untrusted) compute node returns new triples along with the triples it used to derive them, it is very cheap to verify that this inference is indeed true: it is enough to verify that the inference is indeed correct and that the triple used indeed exist in the input data. The computationally expensive part (i.e. indexing and finding the triple combinations that produce inference) is performed by the client nodes.

- The output data is less than the input data. Clients should only return new knowledge. This aligns very well with the characteristics of current consumer-oriented Internet connections (e.g. DSL and Cable), where download speed is an order of magnitude faster than upload speed.

The approach in chapter 8 is well-suited for the implementation of the server backbone of this grid because:

- The application is very data intensive. Thus, several servers are required. Furthermore, these servers can be geographically distributed, or even be within LANs, for organizations with many clients

- Server nodes are autonomous. Even if the server network gets partitioned or the connections between servers become slow, computation will continue with existing data until connectivity returns to normal

The main challenge that needs to be overcome is efficiently determining whether a triple exists. We expect that the exit-door mechanism that we have developed in
Marvin (chapter 8) can play a dual role: detection and elimination of duplicates and verification of the existence of triples.

10.1.3 Private vs public data

Traditionally, research and industry has focused on some of the goals identified in the introduction.

In the area of the Semantic Web, focus was given on consistency (and inconsistency), openness and interoperability. Control has been relatively neglected. Computational scalability has become an issue relatively recently, with the emergence of large datasets.

In search engines, the focus has been on scalability and completeness with practically no effort on control, openness and interoperability.

Web 2.0 applications have given some focus on control, in the sense that publishers may define access rights and may modify content. Attention was also given in interoperability in the field of Web Services.

We claim that a more holistic approach to the design of systems to network data is needed, taking into account the five goals set in the introduction. Recognizing that we live in an imperfect world and have to live with imperfect systems, we provide a view about what can be sacrificed and why.

We start with the observation that some of the goals apply to some kinds of data more than others:

- For personal data, the most important goals are control, completeness and openness. Users expect that they always have access to all of their own data and that their data is only accessible by them. Needless to say, openness to participation is also a necessity.

- For public data, it is not expected that all data is accessible: failures and incompleteness are acceptable. On the other hand, scalability, interoperability and openness to protocols are imperative.

Systems could adapt their behavior to the characteristics of the data they store. An interesting research topic would be how to combine large, public and trusted data with private and untrusted data. The fact that the former is large and will be used in multiple occasions indicate that a forward reasoning approach would perform better while the fact that the latter is untrusted prohibits this.

A system could combine forward and backward reasoning as follows: Use a forward-reasoning approach (like the one presented in chapter 8) to calculate the complete inference of a set of public and fairly reliable data (for example linked open data). For private data, use a backward reasoning approach (such as the one in chapter 7) which also guarantees control (for example, using the privacy-enabled sharing presented in chapter 6). Note that public and private data cross-reference
each-other, thus the two mechanisms are not independent. Furthermore, the computational resources required for the forward reasoning can be offered by the servers maintaining the (private) data for the backward reasoning.

Generally, it would be desirable to move the public data to the private data, since the latter should not be entrusted to other nodes. Since the public data is large, an efficient selection mechanism is required. In addition, the queries for public data can give out information about the private data, which should also be considered. Developing systems that combine private and public as well as trusted and untrusted data would be an interesting research direction.

10.2 Outlook

There are several problems and research domains adjacent to the methods we have developed that were not covered in this thesis, yet play a role in the development of scalable discovery systems.

We have mainly dealt with how to find data and what to do with it, not examining which and whose data to use. It is important to be able to detect the most important information. The notion of importance may have several meanings: it can be data that is most often requested (relevant for caching techniques), data that can be used to infer new data (relevant for reasoning), data that is the most useful (relevant for information retrieval), data that is the most useful to a specific user (relevant for personalization), data that comes from some source (relevant for trust) or data that is highly rated (relevant for reputation).

These fields are symbiotic with discovery. For example, the indexing mechanisms of a discovery system can be used to give an estimate of data popularity, which in turn can be used for information retrieval while information retrieval techniques can provide useful statistical information for indexing. As a second example, personalization techniques could give feedback to caching-like techniques to proactively replicate content.

When combining information, questions arise about the trust and reputation of the inferred data and the access control policies. Can it be that a user has access to inferred information but not to the information that was used to make the inference? What is the aggregate trust and reputation?

Our research also has impact on other fields:

- The privacy-preserving discovery method presented in chapter 3 and chapter 6 allows a more open set of privacy and access control policies. Typically, in privacy preserving discovery [134, 135], a third party enforces access control and privacy policies. This places limitations on the policies that can be used (i.e. it should be possible that they are enforced remotely) and requires that the the third party is entrusted both with the policies themselves and their enforcement.
10.2. OUTLOOK

- The divide-conquer-swap strategy developed in chapter 8 can be used to solve other problems. Our distributed reasoning approach searched for “combinations” of Semantic Web data to do reasoning. We could use the same implementation to search for “combinations” of genes to do evolutionary computation.

- A combination of the distributed backward reasoning approach in 7 and the privacy-preserving sharing approach in 6 can be used to design a global virtual RDF repository with transparent data access on web scale. That would, in turn, have impact on query languages and access control.

- The scalable rendez-vous method in chapter 4 can be applied in networks of nodes with limited connectivity and capabilities (for example sensor networks), since is poses few restrictions on network topology and is very lightweight.

In this thesis, we have presented algorithms, infrastructures and an application for scalable distributed discovery. We have started off with a set of goals and developed systems that fulfill them. Our findings indicate that it is possible to design scalable discovery systems using a peer-to-peer infrastructure. Nevertheless, sacrifices have to be made in some goals to satisfy the rest.

Our research was of an exploratory nature, mapping a segment of the design space. Interesting future work lies in further exploring this space. This exploration should be done both in the space of the five goals we have defined in the introduction and the type of discovery system. Thus, an additional dimension should be added to table 10.1 to represent the type of resources in the system and further experimentation should be done to map the complete space.

To conclude, we believe that a fertile research field lies in web-scale discovery systems that can make the right sacrifices, adapting their behavior to the characteristics of the data they manage. In this thesis, we have conducted exploratory research through the design of a series of algorithms and systems. We have identified some trade-offs and set the ground for future research.
Nederlandse samenvatting

Het Web is het meest gebruikte data netwerk van dit moment. Initieel werd het Web voornamelijk gebruikt voor het ophalen van documenten. Echter, met de opkomst van Web 2.0, het Uniquitous Web, het Semantic Web en de vele apparaten die Web content genereren, is het Web veel persoonlijker, semantisch en dynamisch geworden.

Deze ontwikkelingen stellen nieuwe eisen aan het Web op het gebied van Schaalbaarheid, interoperabiliteit, openheid, controle en compleetheid. Om aan deze eisen te kunnen voldoen, hebben wij een serie methoden ontwikkeld:

Allereerst hebben wij een methode laten zien die de prestaties van DHT-gebaseerde indexen verbetert op basis van populariteit van termen (hoofdstuk 2). Deze methode detecteert populaire termen, en voorkomt dat deze geïndexeerd worden, waarbij compleetheid wordt ingevuld voor schaalbaarheid. Daarbij wordt de doelstelling van open participatie ondersteund, doordat elke host tot het onderliggende peer-to-peer netwerk kan toetreden.

Vervolgens hebben we een methode ontwikkeld om versleutelde, gedistribueerde indexen te kunnen benutten, welke gebruik maken van afgekorte, cryptografische hashes (hoofdstuk 3). Net als de vorige methode draagt ook deze index bij aan open participatie op het web. Tevens biedt het extra controle over gegevens doordat het mogelijk is informatie te beschikbaar te maken, zonder direct de exacte inhoud te hoeven publiceren.

Als derde werd in hoofdstuk 4 een methode beschreven voor een schaalbaar rendez-vous mechanisme. Het combineert de deterministische eigenschappen van een DHT-achtig systeem, met een load balanancing mechanisme op basis van willekeurige uitwisseling van gegevens tussen de knopen. Hierdoor worden de prestaties verhoogd. Deze methode garandeert op lange termijn compleetheid van het resultaat. Aangezien door deze uitwisseling de gegevens echter niet meer alleen bij de eigenaar liggen, is het nodig dat de eigenaar het mechanisme moet vertrouwen. De controle over de gegevens neemt dus wel af met deze methode.

In hoofdstuk 5 werd vervolgens een schaalbare infrastructuur voor webservices gepresenteerd. In het algemeen gesproken kan gesteld worden dat Web-services zich richten op interoperabiliteit: zowel publicatie als het gebruik zijn open.

Ten vijfde is een scenario beschreven om met behoud van de privacy informatie te kunnen delen (hoofdstuk 6). De eerder beschreven schaalbare gedistribueerde index
wordt hierbij gecombineerd met een anoniem netwerk om de privacy te garanderen. Er worden uitsluitend betekenisloze identifiers gepubliceerd, en onderhandelingen over toegang tot de gegeven vinden plaats op peer-to-peer niveau, zodat de volledige controle lokaal kan worden gedefinieerd en gecontroleerd. Tevens heeft dit meer interoperabiliteit en openheid tot gevolg.

Ten zesde werd in hoofdstuk 7 een backward-reasoning methode op peer-to-peer basis geintroduceerd. Gegevens worden lokaal opgeslagen, wat de controle over de gegevens ten goede komt, en in een DHT geïndexeerd, wat gunstig is voor de schaalbaarheid. Net als bij de eerder beschreven methoden, wordt de toegang tot de gegevens lokaal geregeld, wat zorgt voor openheid en interoperabiliteit.

In hoofdstuk 8 werd een forward reasoning methode getoond die gebruik maakt van het rendez-vous mechanisme uit hoofdstuk 4. Hierdoor deelt het de nadelen ten aanzien van de controle over de gegeven, maar is het resultaat uiteindelijk wel compleet. De methode maakt het mogelijk verschillende reasoners te gebruiken, wat de openheid en interoperabiliteit ten goede komt.

Ten slotte richt het OpenKnowledge systeem, dat in hoofdstuk 9 werd beschreven, zich op het schaalbaar delen van informatie door middel van een open interacties tussen peers. Dit systeem maakt interoperabiliteit en controle mogelijk door gebruik te maken van geformaliseerde interactie modellen.

Hoewel sommige methoden aan meerdere eisen kan voldoen, is gebleken dat geen enkele methode in staat is aan alle eisen te voldoen. Voor de toekomst is verder onderzoek gewenst naar de omstandigheden waarin de verschillende eisen van critisch belang zijn.
Bibliography


[70] E. Liarou, S. Idreos, and M. Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In I. Cruz, S. Decker,


[128] Universal Description, Discovery and Integration of Business for the Web.


SIKS Dissertation Series

1998

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Nick J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennisstechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analysis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====

2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA)
Learning as problem solving
2001-4 Evgeni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9 Pieter Jan ‘t Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====

2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
Applied legal epistemology: Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel (KUB)
   Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
   Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
   Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (UvA)
   Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
   A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wicke de Vries (UU)
   Agent Interaction: Abstract Approaches to Modelling, Programming and 
   Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
   Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
   The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
   Understanding, Modeling, and Improving Main-Memory Database Performance

2003-01 Heiner Stuckenschmidt (VU)
   Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
   Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
   Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
   Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
   Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
   Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
   Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
   Repair Based Scheduling

2003-09 Rens Kortmann (UM)
   The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
   Electronic Business Negotiation: Some experimental studies on the interaction 
   between medium, innovation context and culture

2003-11 Simon Keizer (UT)
   Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
   Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
   Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
   Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
   Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
   Feature Grammar Systems - Incremental Maintenance of Indexes to 
   Digital Media Warehouses

2003-17 David Jansen (UT)
   Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
   Learning Search Decisions
2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martín Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjens (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI
2005-07 Flavius Frasincar (TUE)  
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 RichardVDovjak (TUE)  
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)  
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 AndersBouwer (UVA)  
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 ElthOgston (VU)  
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 CsabaBoer (EUR)  
Distributed Simulation in Industry

2005-13 FredHamburg (UL)  
Een Computermode voor het Ondersteunen van Euthanasiebeslissingen

2005-14 BorysOmelayenko (VU)  
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 TiborBosse (VU)  
Analysis of the Dynamics of Cognitive Processes

2005-16 JorisGraaumans (UU)  
Usability of XML Query Languages

2005-17 BorisShishkov (TUD)  
Software Specification Based on Re-usable Business Components

2006-01 SamuilAngelov (TUE)  
Foundations of B2B Electronic Contracting

2006-02 CristinaChisalita (VU)  
Contextual issues in the design and use of information technology in organizations

2006-03 NoorChristoph (UVA)  
The role of metacognitive skills in learning to solve problems

2006-04 MartaSabou (VU)  
Building Web Service Ontologies

2006-05 CeesPierik (UU)  
Validation Techniques for Object-Oriented Proof Outlines

2006-06 ZivBaida (VU)  
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 MarkoSmiljanic (UT)  
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 ElcoHerder (UT)  
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 MohamedWaldan (UM)  
Automatic Formulation of the Auditor’s Opinion

2006-10 RonnySiebes (VU)  
Semantic Routing in Peer-to-Peer Systems

2006-11 JoeriVanRuth (UT)  
Flattening Queries over Nested Data Types

2006-12 BertBongers (VU)  
Interactivation - Towards an e-ology of people, our technological environment, and the arts

2006-13 Henk-JanLebbink (UU)  
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
  Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
  CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
  Approximation Methods for Efficient Learning of Bayesian Networks
2006-17
Stacey Nagata (UU)
  User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhikun (UVA)
  Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
  Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
  Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
  Aptness on the Web
2006-22 Paul de Vrieze (RUN)
  Fundamentals of Adaptive Personalisation
2006-23 Ion Juvina (UU)
  Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
  Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
  Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlovic (UT)
  Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
  Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA)
  Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT)
  Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
  Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
  Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
  Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
  Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA)
  Applied Text Analytics for Blogs
2007-07 Natasa Jovanovic (UT)
  To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)
  Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU)
  Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU)
  Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11 Natalia Stash (TUE)
  Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN)
  Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations
2007-19 David Levy (UM)
Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
2007-21 Karrianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns
2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems
2007-24 Georgina Ramirez Camps (CWI)
Structural Features in XML Retrieval
2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====
2008====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
2008-02 Alexei Sharpaniskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations
2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach
2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration
2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning
2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
2008-10 Wauter Bosma (UT)
Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort 2008-15 Martijn van Otterlo (UT)
2008-16 Henriette van Vugt (VU)  
Embodied agents from a user’s perspective

2008-17 Martin Op’t Land (TUD)  
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)  
Adaptive Active Vision

2008-19 Henning Rode (UT)  
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)  
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)  
People Search in the Enterprise

2008-22 Henk Koning (UU)  
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)  
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zarko Aleksovski (VU)  
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)  
Efficient and Equitable Exchange in Air Traffic Management

2008-26 Marijn Huijbregts (UT)  
Plan Repair using Spender-signed Currency

2008-27 Hubert Vogten (OU)  
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-28 Ildiko Flesch (RUN)  
Design and Implementation Strategies for IMS Learning Design

2008-29 Dennis Reidsma (UT)  
On the Use of Independence Relations in Bayesian Networks

2008-30 Marrie van Atteveldt (VU)  
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)  
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)  
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)  
Scientific Workflow Design: theoretical and practical issues

2008-34 Jeroen de Knijf (UU)  
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)  
Dendritic morphologies: function shapes structure

== ==
2009
== ==

2009-01 Rasa Jurgelenaite (RUN)  
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)  
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)  
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)  
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)  
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)  
Understanding Classification
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Masuthe (TUE, Humboldt-Universitaet zu Berlin)
perating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Malesym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess 2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data 2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controlling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelwijnt (UL)
Privacy en Politiegerechts; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
A Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models