

# Massively Scalable Web Service Discovery

George Anadiotis, Spyros Kotoulas, Holger Lausen, Ronny Siebes\*

## Abstract

*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 paper 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.*

## 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.”<sup>1</sup> 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 [2]. 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.

\*George Anadiotis, Spyros Kotoulas and Ronny Siebes are with the Department of Computer Science, VU university, Amsterdam, the Netherlands. Holger Lausen is with seekda, a spin-off of the University of Innsbruck, Semantic Technology Institute (STI) Innsbruck. This work was partially supported by the EU OpenKnowledge project (FP6-027253).

<sup>1</sup>Computer Weekly, March 11, 2003 (<http://www.computerweekly.com/Articles/2003/03/11/193049/forrester-ceo-forecasts-web-services-storm.htm>)

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 on a-priori semantic agreement, which is usually expressed as a shared ontology or a categorisation [18, 12, 10, 17] 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 paper 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<sup>2</sup>, 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.

## 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.

<sup>2</sup><http://www.seekda.com/>

## 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 overlaid 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.

## 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 [9]. 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 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: *store(key, object)*, that stores an object identified by its key, and *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.

## 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<sup>3</sup>. A naive approach for multiple

<sup>3</sup>disjunctive queries are easier to support as they can be split into a set of single-term queries

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 [1]. 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.
- *Load distribution.* The fact that term frequency distribution usually follows a Zipf pattern can cause severe load distribution problems, since the bins (ie. 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 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.

### 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 previous work on rarity-based routing [6]. 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 [16], 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 [8, 5], 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, distributedly 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<sup>4</sup>. 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-

<sup>4</sup>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

**Require:** Let  $\mathcal{D} \subseteq \{t_1, t_2, \dots, t_n\}$  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  $t$ .

```

1: while true do
2:    $t_{sel} := t_m \forall t \in (d - T) |D(p, t)| \geq |D(p, t_m)|$ 
3:    $T := T \cup (t | |D(p, t)| > f_{max}) \cup t_{sel}$ 
4:   if ( $t_{sel} = \emptyset$ ) then
5:     return
6:   else
7:     //Send to the peer responsible for  $t_{sel}$ 
8:      $p := p_{t_{sel}}$ 
9:   end if
10: end while

```

**Require:** A query for terms  $q = \{t_1 \dots t_n\}$ .

```

1: while true do
2:    $t_{sel} := t_m \forall t \in (q - T), |D(p, t)| > |D(p, t_m)|$ 
3:    $T := T \cup t_{sel}$ 
4:   if ( $t_{sel} = \emptyset \vee (query\ satisfied)$ ) then
5:     return
6:   else
7:     //Send to the peer responsible for term
8:      $p := p_{t_{sel}}$ 
9:   end if
10: end while

```

#### Algorithm 1. Rarity-based walk

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 2, the description for an example Stock Quote<sup>5</sup> 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.

<sup>5</sup><http://www.xignite.com/xQuotes.asmx?WSDL>

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').

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

### 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  $f_{max}$  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.

### 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  $f_{max}$  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 [15]. 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 than the value. For example the query

```
(message_name='`tickerSymbol`',
message_element='`string`',
port_name='`StockQuotePort`',
binding='`StockQuoteBinding`')
```

would be translated to

```
(message_name, tickerSymbol,
message_element, binding,
string,port_name, StockQuotePort,
StockQuoteBinding).
```

## 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.

### 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<sup>6</sup> Java API, but no further filtering was performed (such as eliminating

<sup>6</sup><http://sourceforge.net/projects/wsdl4j>

duplicates or verifying the availability of the service implementation).

The next step after crawling was parsing the resulting WSDL data. Standard parsers (e.g. tsearch2<sup>7</sup>) 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<sup>8</sup>, without which a standard parser does not extract any terms at all.

In order to increase the number of descriptive terms associated with a WSDL document, we developed and use our own specialized parser and tokenizer, that also take into account the semantics of WSDL documents such as name attributes of service and portTypes. Attribute name in WSDL documents are very expressive and contain term 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, 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 term “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, 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.

## 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. 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

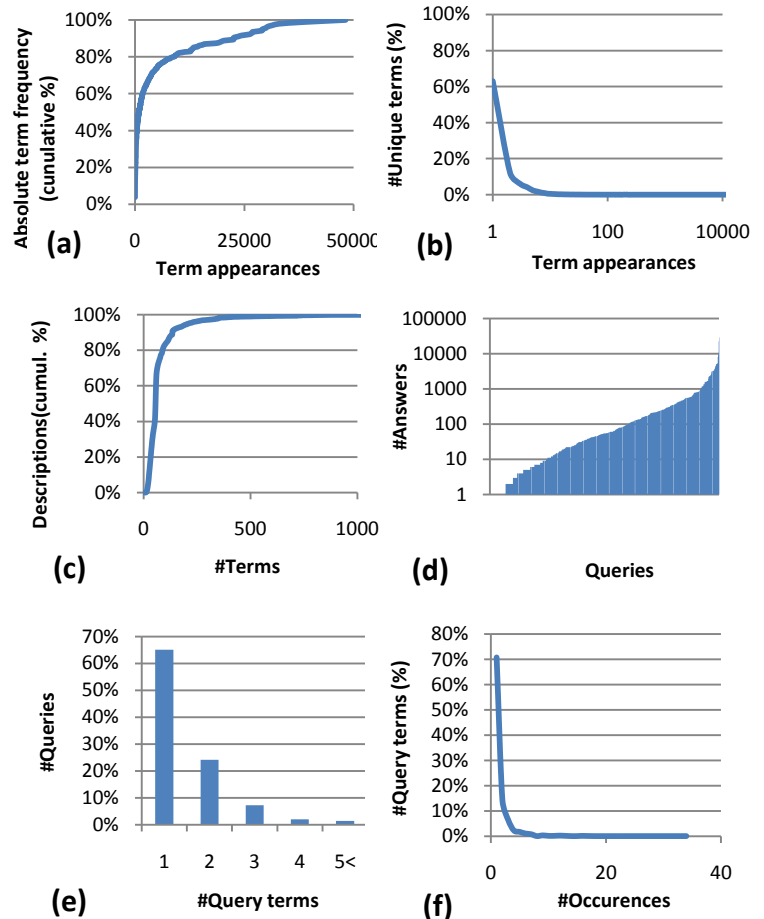


Figure 1. Dataset statistics

<sup>7</sup><http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/>

<sup>8</sup>based on seekda’s database analysis

Fig. 1(a) (absolute term frequency by term appearances, expressed as cumulative percentage), and Fig. 1(b) (number of unique terms by term appearances). Fig. 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. 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 distribution is displayed in Fig. 1(f), from which we can see that 70% of query terms appear only once. Fig. 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. 1(d) we conclude that most queries have between 10 and 100 answers.

### 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 paper we made  $N=25$  as a default value. Therefore, we define recall as follows:

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

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.

### 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 algo-

ritms described in 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<sup>9</sup>. The search primitives of the middle layer were provided by the implementation of the algorithm in Section 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 [14].

### 4.5 Large-scale emulation

We tested and evaluated our system on the DAS-3 distributed supercomputer<sup>10</sup>, 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<sup>11</sup> 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 4.7. Results were written to files and processed off-line, so as to minimize the intrusiveness of the evaluation.

### 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).

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

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

<sup>11</sup><http://www.globus.org/>

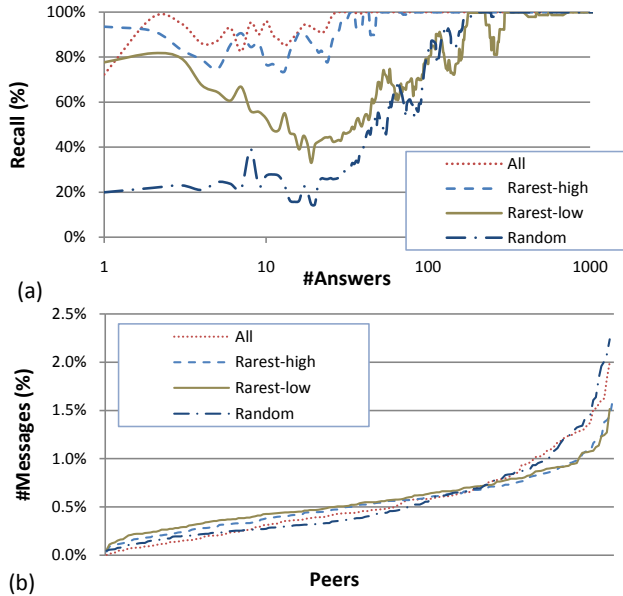


Figure 2. Experimental results

**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 meant to 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 using less messages.

## 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 1 shows our most important findings: Replicating indexes according to all terms<sup>12</sup> gave comparable recall to the rarity-based walk with parameter  $f_{max} = 20$  (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-

<sup>12</sup>note that the number of messages used is 66.97 instead of the theoretical 76 because of peer failures and lost messages

based walk with parameter  $f_{max} = 3$  outperforms replication according to a set of terms chosen randomly, yielding 29% higher recall with roughly the same number of messages.

Figure 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. 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 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.

## 5 Related work

A first group of systems we will examine are attempts to directly extend UDDI by applying some sort of distributed

Setting	Recall	#mess./descr.
All	94.67%	66.97
Rarest-high	93.45%	17.19
Rarest-low	74.04%	9.22
Random	57.21%	9.75

**Table 1. Summary of experimental results**

techniques on top of it.

In [11], 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 [3]. 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 [12]. 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 [10], which uses DAML-S and Gnutella as the ontology and P2P substrates correspondingly. A similar approach is [17], 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 [18]. 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 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 [13] 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 [4], 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 [7], 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.

## 6 Conclusions and Future Work

In this paper 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.

## 7 Acknowledgements

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

## References

- [1] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. INFOCOM*, New York, USA, 1999.
- [2] Clement L., Hatley A., Riegen C.v. and Rogers T. Universal Description, Discovery and Integration (UDDI) 3.0.2, 2004.
- [3] Z. Du, J. Huai, and Y. Liu. Ad-UDDI: An Active and Distributed Service Registry. In *Proc. TES*, 2005.
- [4] F. Emekci, O. D. Sahin, D. Agrawal, and A. E. Abbadi. A Peer-to-Peer Framework for Web Service Discovery with Ranking. In *Proc. ICWS*, Washington, DC, USA, 2004.
- [5] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. VLDB*, 2003.
- [6] S. Kotoulas and R. Siebes. Rarity-based routing in peer-to-peer networks. In *COPS Workshop at IEEE WETICE*, Paris, France, 2007.
- [7] M. Koubarakis, I. Miliaraki, Z. Kaoudi, M. Magiridou, and A. Papadakis-Pesaresi. Semantic Grid Resource Discovery using DHTs in Atlas. In *Proc. GGF Semantic Grid Workshop*, Athens, Greece, 2006.
- [8] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid p2p search infrastructure. In *Proc. IPTPS*, 2004.
- [9] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2004.
- [10] M. Paolucci, K. P. Sycara, T. Nishimura, and N. Srinivasan. Using DAML-S for P2P Discovery. In L.-J. Zhang, editor, *ICWS*, pages 203–207. CSREA Press, 2003.
- [11] M. Papazoglou, J. Jang, and B. Kraemer. Leveraging web-services and peer-to-peer networks. In *Proc. CAiSE*, 2002.
- [12] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. In *Proc. P2P*, Washington, DC, USA, 2002.
- [13] C. Schmidt and M. Parashar. A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web*, 7(2):211–229, 2004.
- [14] R. Siebes, D. Dupplaw, S. Kotoulas, A. Perreau de Pinninck Bas, F. van Harmelen, and D. Robertson. The OpenKnowledge System: an interaction-centered approach to knowledge sharing. In *Proc. CoopIS*, 2007.
- [15] G. Skobeltsyn and K. Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *Proc. P2PIR*, New York, NY, USA, 2006. ACM Press.
- [16] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *Proc. IEEE INFOCOM*, San Francisco, CA, USA, 2003.
- [17] I. Toma, B. Sapkota, J. Scicluna, J. M. Gomez, D. Roman, and D. Fensel. A P2P Discovery Mechanism for Web Service Execution Environment. In *Proc. WIW05*, Innsbruck, Austria, 2005.
- [18] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Inf. Technol. and Management*, 6(1):17–39, 2005.