ABSTRACT

Peer-to-peer networks have recently gained importance as environments for digital libraries. In this paper, we propose a service-oriented architecture for supporting two important access modes, searching and browsing, in a hierarchical network. Service-oriented architectures allow for loosely-coupled systems, and the combination of multiple services enables integration of the two paradigms. The services have been designed for optimised reusability and extensibility. E.g., distributed searching and browsing both use the same statistical metadata, and thus the same storage service, for solving their tasks. Browsing services provide an overview over the available content, and interactive exploration of relevant portions of the data space. Resource selection services improve the efficiency of the search process by cost-optimised query routing, and schema mapping services facilitate the integration of data from heterogeneous schemas into one distributed digital library.

1. INTRODUCTION

Literature search is one of the major work tasks for scientists: Sometimes, a particular paper has to be found, for which e.g. the titles and the authors are already known. In other cases, a researcher needs to get an overview over the state-of-the-art in a particular field. Two different paradigms for supporting such search tasks can be distinguished: For retrieval, the user needs some knowledge which is put into the form of a query, and relevant documents are returned by the system. For browsing, in contrast, the system presents the collection in such a way that a user can easily get an overview and find useful documents, without the need of explicitly specifying the vague information need.

Typically, multiple sources (called digital libraries, DLs) have to be combined for satisfying an information need, like the ACM digital libraries or personal collections with technical reports. Searching in such a distributed environment is cumbersome and frustrating. A user wants a single-stop solution which efficiently supports the search process in the underlying libraries and solves the problems arising from heterogeneous schemas (e.g. BibTeX, Dublin Core, MARC).

Thus, systems have to deal with different information access strategies, distributed collections and heterogeneous documents.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGIR '05 Workshop on Heterogeneous and Distributed Information Retrieval, August 19, 2005, Salvador, Brazil
Copyright 2005 ACM X-XXXX-XX-XXX/XX ...$5.00.

This paper proposes a peer-to-peer network of heterogeneous, reusable services, which is currently implemented in the Pepper prototype\cite{Pepper}. Peer-to-peer networks are well known from the context of recreational file-sharing applications like Gnutella and KaZaA, but have recently obtained broader interest within the scientific community as well. In our system, each user can autonomously maintain her own collection of documents which forms the basis of retrieval and browsing. This structure makes it particularly easy to provide e.g. fulltexts (as PDFs) of own publications or metadata collections (e.g. in BibTeX style) to the public. Other peers can wrap web portals like ACM. Special services deal with the heterogeneity of collections by transforming queries or documents.

Basically, a service provides a modular piece of functionality which can be invoked using standard protocols (e.g. SOAP). A service can be reused in different applications or for different scenarios: For example, the task of storing and providing statistics about document collections, or the task of transforming documents into different schemas are required for both retrieval and browsing. The Pepper system integrates about a dozen services which facilitate searching, browsing, or both. For efficiency, the design is based on a hierarchical network, where the peers are divided into two distinct classes, i.e. hubs (directory server peers) and leaves (typically running on a user machine).

The following section describes related work in the fields of architectures and distributed retrieval and browsing. The proposed architecture and the set of services are described in section 3. In section 4, we elaborate on the concept and generation of resource descriptions which are needed for most of the services. The problems of efficient query routing in a distributed document collection, and of providing browsing functionality, are discussed in sections 5 and 6. In section 7, we tackle the issue of integrating data from heterogeneous schemas, and in section 8, we wrap up with a conclusion and an outlook, especially on future extensions of the network.

2. RELATED WORK

This section reflects research which is related to this paper w.r.t. to the architecture or its major components (resource selection and browsing).

2.1 Architectures

Typically, three different topologies are distinguished for peer-to-peer networks: unstructured, structured and hierarchical networks. All peers are equal in unstructured networks, and no restrictions are imposed on the topology. Structured networks, on the other hand, have a regular topology, e.g. cycles or cubes (see below). Hierarchical networks partition the peers into two sets.

\cite{Pepper http://www.pepper-project.org}
Hubs are dedicated servers with large memory and high computation power. They are e.g. responsible for the routing of queries (i.e. resource selection). Leaf peers, on the other hand, host e.g. digital libraries or other services like document transformation services. They do not forward the query any further. As an advantage of this topology, less peers are involved in the query routing process, thus potentially reducing the number of messages.

Service-oriented architectures (SOA) become more and more popular in distributed environments, and recently gained importance in the field of digital libraries and peer-to-peer networks.

Furmento et al. [14] present a general service-oriented architecture where the same service implementation can be used on top of different transport layers, including Jini, JXTA and the Open Grid Service Infrastructure (OGSI). They claim that the major problems with JXTA are lack of security and—more important for digital libraries—performance, and recommend the usage of Jini.

A combination of SOA, peer-to-peer networks, and ontologies for distributed knowledge management is proposed in [15]. Different services like knowledge base manager, service planning services, semantic registries or user interface components are provided in a hierarchical network.

A combination of peer-to-peer networks (JXTA), service-oriented architectures, and digital libraries is described in [8]. Their system encompasses many components which we also develop for Pepper, like query and document transformation services for handling heterogeneous services, or the dynamic selection of services based on semantic service descriptions via OWL-S. However, they use distributed hash tables, and a clustering of peers is used for finding the most appropriate search services, instead of hierarchical networks and IR techniques for resource selection.

### 2.2 IR-based resource selection

Resource selection is the task to decide to which digital libraries (or peers) a query should be broadcasted, and how many documents have to be retrieved from each selected DL. This problem has been investigated for federated digital libraries for about a decade. Federated digital libraries have a much simpler topology than peer-to-peer networks: All underlying libraries are directly connected to a single centralised component which is responsible for the selection process; in contrast, resource selection is decentralised in peer-to-peer networks.

Different to the decision-theoretic framework employed in Pepper, most of the other selection algorithms rank the digital libraries (DLs) w.r.t. their similarity to the query, and retrieve a constant number of documents from the top-ranked libraries. In a final step, the results from the selected libraries are combined in result merging (often also called “data fusion”).

The popular CORI [3] approach is based on the INQUERY retrieval system. The resource selection task is similar to document retrieval, where documents are replaced by collections (considered as the concatenation of all documents in the collection). CORI also covers the data fusion problem, where the library score is used to normalise the document score.

Another ranking approach is based on language models [27]. Basically, the language model of the collection is smoothed with a collection-independent (system-wide) language model, and KL-divergence is used for ranking the DLs. The final document ranking is computed in a result merging step by using the original (collection-biased) document probabilities, the DL scores, a smoothing factor, and Bayesian inversion. The quality of this approach is slightly better than CORI.

The language model approach has been extended towards hierarchical peer-to-peer networks in [19] for ranking neighbour peers (leaves and hubs). Hub descriptions describe its neighbourhood (which is not limited to the directly connected DLs), where the influence of term frequencies of distant DLs is exponentially decreased. Leaves and hubs are selected separately, as leaf and hub descriptions are not in the same order of magnitude. A fixed number of hubs is selected, while unsupervised learning is employed for computing a threshold for the number of selected leaves.

All of these IR-based resource selection techniques require so-called “resource descriptions”. A resource description contains statistical metadata about the collection, e.g. document frequencies for CORI or a collection language model. Resource descriptions can be provided by the collection itself, or discovered via query-based sampling [2].

### 2.3 Database-oriented query routing in peer-to-peer networks

Research on peer-to-peer networks has mostly focused on database-oriented approaches for sending (“routing”) a query through the network. Similar to the resource selection strategies described above, query routing aims at minimising the number of hops for answering a query while yielding good results; however, they typically do not compute the degree of usefulness of a resource like traditional resource selection techniques.

One of the early peer-to-peer networks is Gnutella, which sends queries to all known neighbour peers (flooding technique). This approach imposes too high a load onto the network, and, thus, does not scale [26].

The effect of flooding can be reduced in hierarchical networks with hubs (sometimes also called super nodes or super peers). E.g., KaZaA forwards queries to other super nodes or answers them based on up-to-date peer index copies. Obviously, using index copies of potentially large, complex DLs is undesirable. Another example of hierarchical networks is JXTA Search [29], where hubs select those neighbours whose capabilities match the query. Cooperation between hubs is intended but not yet defined.

Routing indices have been proposed for deciding to which neighbours a query should be sent [4]. A routing index stores the number of documents per topic (or term) which can be accessed via each neighbour peer. So, structurally they are similar to resource descriptions; the major difference is that resource descriptions aim at finding useful (“relevant”) documents, while routing indices are only suitable for Boolean retrieval.

PlanetP [5] combines Bloom filters and the vector-space IR model. In a first step, all known peers are ranked according to the number of query terms they contain. For improved efficiency, Bloom filters are employed. A Bloom filter is a bit array; n different hash functions are applied to the term for computing n indices, and the bits in every index position are set to one. The Bloom filters are flooded through the network (“gossiping”), so that every peer basically knows all other peers. In a second step, the top-ranked peers are queried, and the vector-space model is employed for retrieving and ranking documents. As in the case of routing indices, PlanetP does not consider the quality of documents stored in peers. A second major drawback is that each peer has to maintain the Bloom filter of every other peer in the network.

Distributed hash tables (DHT) like Chord [28] or CAN [25] are another popular technique for efficiently finding files or documents (identified by a key) in peer-to-peer networks. In Chord, each peer has an integer address, and is responsible for keys which are mapped onto its address by applying a hash function. The addresses induce a cycle of peers; as shortcuts, each peer also stores pointers
to other peers in exponentially increasing distances. The exponentially increasing distances of the peer pointers ensure that the peer is found after $O(\log n)$ hops.

RDF-based approaches for query routing are Edutella [20] and the extension OAI-P2P [1]; both support only exact match.

2.4 Cluster-based browsing

The browsing method which we propose for our peer-to-peer architecture is an adaptation of pre-processed Scatter/Gather [6]. In each step of a Scatter/Gather [7] browsing session, the user is presented with a cluster-based partitioning of the documents which she selected for browsing. From these clusters, she chooses a subset of interest, which is clustered again (in more detail) and re-presented to the user, and so on. In pre-processed Scatter/Gather, a cluster hierarchy is computed offline, and most of the browsing and online clustering then uses the cluster descriptions, instead of the contained documents. To the best of our knowledge, Scatter/Gather has not previously been generalised to the highly distributed environment of peer-to-peer networks.

Content-based clustering as such, on the other hand, has already been employed in peer-to-peer networks as well. In [11], a distributed version of the k-means algorithm is performed in a peer-to-peer network, profiting from the parallelisation thus made possible. While the documents do not have to be transferred to one single peer for clustering, cluster centroids are still propagated to all participating peers in each clustering step.

[17] applies clustering on two levels. The documents on each peer are clustered in order to find the topics present on the peer. On joining the network, the peers themselves are assigned into context-aware groups (CAGs), which form an overlapping clustering on the basis of the topics on the peers. The centroids of the resulting CAGs are then used for query routing. Both the clustering of peers into CAGs, and the query routing are tasks performed by hubs which have to maintain a list of all CAG centroids.

Both approaches use a local clustering of documents as well as a global list of cluster centroids. Yet, neither one addresses browsing. In section 6, we will follow similar principles to create a distributed, two-level cluster hierarchy for Scatter/Gather browsing.

3. A SERVICE-ORIENTED PEER-TO-PEER NETWORK

This section describes the architecture of the Pepper prototype, which is build on the ideas of service-oriented architectures and peer-to-peer networks. The system is built upon Sun’s JXTA [16], one of the state-of-the-art peer-to-peer frameworks. The major goal of JXTA is to define a set of open, XML-based protocols for connecting a variety of devices, e.g. PCs, PDAs or mobile phones. The JXTA reference implementation is written in Java.4

3.1 Network topology

Pepper borrows an important idea from other peer-to-peer networks like KaZaA or JXTA Search [29]: The set of peers is partitioned into two disjoint classes, hubs (sometimes also called super peers or ultra peers) and leaves (also called DL peers):

Hubs are static dedicated peers with large memory and high computation power, which remain online for most of the time. Hubs work as directory peers, which means that other hubs or leaves register (some of) their services at a hub. As a consequence, hubs are e.g. responsible for query routing.

Leaves host most of the services. E.g. if a user wants to share documents with others in the network, a search service is running on his PC which allows for retrieval on that local collection (on that “digital library”). Leaves are only connected to hubs; there is no direct connection between leaf peers.

The advantage of this topology (called hierarchical network) is the smaller number of peers that are involved in the query routing process, and the lower number of messages required. Furthermore, the network diameter is reduced. Together, hierarchical networks scale better than unstructured networks.

3.2 Search and browse services

The Pepper prototype implements a service-oriented architecture, which is based on JXTA SOAP5, a sub-project of JXTA which aims at integrating web services and peer-to-peer networks. Several independent services can be provided by a single peer, which can be retrieved by name. It is particularly easy to convert a conventional web service into a JXTA SOAP service and vice versa. When another peer wants to call such a service, it has to retrieve the service advertisement, send a SOAP message5 to the desired method within the service (“remote procedure calls”), and receive the results.

The Pepper system hosts a heterogeneous set of 11 services. Each service type has its own service interface which defines the methods which are supported. Different implementations of the service interface are possible, e.g. for supporting different routing and retrieval strategies.

A leaf inventory service stores the documents of a particular collection, indexes them, and provides additional metadata about the collection which can be used for retrieval and browsing. A hub inventory service serves a similar function on the hubs. Search services can answer queries, by either utilising a digital library (on the leaf peer) or employing distributed IR techniques (on hubs, see below). Resource selection services (which reside on the hubs) assist in query routing by selecting those search services which seem to be the most appropriate ones, while result merging services compute a single ranked list from multiple lists. Document browsing is supported by dedicated services on the leaf, in interaction with the inventory services on both hubs and leaves. Finally, schema mapping services bridge the gap between heterogeneous schemas. In conclusion, it is easy to integrate (new) services into the network, and many services are reused for different scenarios.

4. RESOURCE DESCRIPTIONS

Both retrieval and browsing require statistical metadata about the collections involved, which is called “resource description”. In Pepper, we consider resource descriptions of different granularity.

4.1 Resource description content

Resource descriptions in Pepper differ from other approaches, e.g. CORI (terms and their document frequencies) or the language model approach (collection language model). In Pepper, a description vector $\vec{o}$ represents an object $o$ as a list of terms with an associated probabilistic weight $\vec{o}(t) \in [0, 1]$. A resource description can consist of a set of description vectors.

In the easiest case, resource descriptions are only available at the document level, i.e. each description vector describes a single document. The term weights in the description vectors are then called indexing weights, and the resource description (a set of document descriptions) is maintained by hubs.

Implementations in C and other programming languages are available, but development is far behind the Java version.

4http://soap.jxta.org/
5http://www.w3.org/TR/SOAP/
description vectors) is equivalent to a document index in information retrieval.

The other extreme is a single description vector for the whole collection. In this case, the vector contains all terms in the collection and the average of the index weights in the single collection $DL$:

$$DL(t) = E(\hat{d}(t)|d \in DL) = \frac{1}{|DL|} \sum_{d \in DL} \hat{d}(t).$$

(1)

Here, $d$ is a document, $\hat{d}$ is its representation with the components $\hat{d}(t)$ for terms $t$, and $|DL|$ denotes the number of documents in the collection.

In a more general setting, the resource description contains $1 \leq n \leq |DL|$ description vectors. Following equation 1, each of the $n$ description vectors contains the averaged term weights of a set of documents. Each vector represents the centroid of a cluster (where a cluster might contain a single document, a set of similar documents, or the whole collection).

For advanced applications like browsing, the resource description contains a hierarchical organisation of vectors (see figure 1), where each vector represents a cluster, and each cluster can combine smaller clusters (up to clusters which consist of a single document, see above). The root of the cluster hierarchy represents the complete document collection. The hierarchy can be computed by any hierarchical clustering algorithm, or by repeated application of a partitioning algorithm, provided that the resulting clusters do not overlap.

For such clusters, equation 1 is still valid, if $DL$ is replaced by the cluster $C$. However, if the cluster is split into sub-clusters $C_1$, $C_2$, $C_3$, for which description vectors $\vec{C}_j$ are already computed, then the vector for $C$ can be computed more efficiently by directly combining (“aggregating”) the description vectors for the $C_j$:

$$\vec{C}(t) = E(\hat{d}(t)|d \in C) = \frac{1}{|C|} \sum_j |C_j| \cdot \vec{C}_j(t).$$

(2)

As a special case, a hub $H$ also stores descriptions of its adjacent hubs $H_i$. Here, each $H_i$ stands for a complete neighbourhood (of leaves $DL_{i,j}$) which can be visited through $H_i$. In a simple yet effective and efficient scenario, this neighbourhood of $H_i$ contains exactly all leaf peers (with their DLs) which are directly connected to $H_i$.

The description of hub $H_i$ is formed by aggregating the resource descriptions of the $DL_{i,j}$ (the search services of those leaves). Thus, hub descriptions can be seen as another level in the resource description hierarchy (on top of DL descriptions), and hub descriptions are computed by aggregating leaf descriptions following equation 2:

$$\vec{H}_i(t) = \frac{1}{|DL_{i,j}|} \sum_j |DL_{i,j}| \cdot \vec{DL}_{i,j}(t).$$

(3)

Thus, hub descriptions can efficiently be constructed, as descriptions of the involved DLs can be easily gathered in the network.

It has been shown that hub-global IDF values outperform local ones [22]. Hub-global IDF values are computed by considering document frequencies not in a single collection but in the combination of all DLs in the neighbourhood of a hub.

Pepper is able to deal with simple schemas. Documents are modelled as a linear list of attributes (like “author”, “editor”, “title” or “year”). A popular example of a schema with such a flat, linear structure is Dublin Core [10]. Such documents can be easily expressed in XML documents (with only two levels of elements), or as binary relations (as used in section 7 for declarative definitions of schema mappings). As a consequence, a resource description in Pepper contains a separate cluster hierarchy for each attribute. For simplicity, we ignore this detail in the following descriptions of retrieval and browsing; both access modes can be extended towards linear schemas in a straightforward way.

4.2 Resource description services

Two closely-related services are responsible for creating, storing and handling resource descriptions in leaves and in hubs:

**Leaf inventory services (LIS)** store the documents in a single collection, the resource description with its clusters and description vectors (including the document index, i.e. the clusters with a single document), and provide full documents and resource descriptions upon request.

**Hub inventory services (HIS)** store resource descriptions of multiple collections (from leaves and hubs) and provide them upon request. HIS are running on hubs.

These resource descriptions can be used by a variety of other services. E.g. the search service uses the single-document description vectors for retrieval on the collection. The resource selection service (see next section) uses the resource description with a single vector for the whole collection, while browsing (see section 6) is based on different levels of the cluster hierarchy.

Peer-to-peer networks are characterised by a dynamic behaviour, where peers appear and disappear at any time. In such an environment, transferring resource descriptions becomes a crucial efficiency issue. Several countermeasures are used in Pepper for improved efficiency:

- Each resource description is quite small, 2MB on average for leaf documents (for about 26,500 documents).
- The resource description can be pruned for resource selection and browsing [18, 7], i.e. only the terms with the highest weights are included.
- The number of resource description transfers is minimised by performing updates offline, and by connecting a leaf to the same hub it has been connected before (if possible).

5. RESOURCE SELECTION

In highly-distributed networks, it is too expensive to send the query to all search services. Resource selection is the task to decide to which peers a query should be broadcasted, and how many documents have to be retrieved from each selected peer.

In the remainder, we assume that a user specifies the number $n$ of documents she wants to receive; this number is sent in combination
with the query \( q \) through the network. The detailed task then is to compute, for every \( DL_i \), the number \( s_i \) of documents which have to be retrieved from \( DL_i \), so that the total number equals a user-specified value \( n \).

### 5.1 Decision-theoretic framework

Traditional resource ranking approaches first find the best collections, and then select a constant number of documents from each selected collection (i.e., \( s_i \in \{0, \text{const}\} \)). In contrast, the decision-theoretic framework (DTF) \[23, 12\] returns those \( s_i \) which optimise the overall selection.

Costs are introduced in DTF as a general-purpose optimisation criteria: E.g., retrieval quality is measured as the number \( r_i(s_i, q) \) of relevant documents in the result set of \( s_i \) documents. Computation time in the DL (which is often assumed to be constant), and communication time for sending the documents (proportional to the number of documents \( s_i \)) leads to an affine linear function. Monetary costs are mainly per-document charges.

Actual costs are unknown at query time, thus expected costs \( EC_i(s_i, q) \) for retrieving \( s_i \) documents have to be considered. They are computed as the weighted sum of the expected costs for the different cost criteria; a user can specify a selection strategy (e.g., “fast results are preferred over high-quality ones”) by weighting factors:

\[
EC_i(s_i, q) := c^{rel} \cdot EC_i(s_i, q)^{rel} + c^{time} \cdot EC_i(s_i, q)^{time} + c^{money} \cdot EC_i(s_i, q)^{money}.
\]

The goal is then to compute a selection vector \( \bar{s} = (s_1, \ldots, s_m) \) which minimises the overall costs:

\[
\bar{s} := \arg\min_{\sum_i s_i = n} \sum_{i=1}^m EC_i(s_i, q).
\]

Retrieval quality, i.e. the expected number \( E[r_i(s_i, q)] \) of relevant documents in the result set of \( s_i \) documents, is estimated in two steps:

- Together with a linear retrieval function, a constant \( c \) for computing probabilities of relevance, query term weights \( \bar{q}(t) \) and resource descriptions, the expected number \( E[rel|q, DL] \) of relevant documents in the whole collection is estimated as:

\[
E[rel|q, DL] = \sum_{d \in DL} Pr[rel|q, d] = |DL| \cdot c \cdot \sum_{t \in \bar{q}(t)} \bar{DL}(d(t)).
\]

- In a second step, an approximation for the recall-precision function is employed for estimating retrieval quality in the top ranks, i.e. for transforming \( E[rel|q, DL] \) onto \( E[r_i(s_i, q)] \).

The search services return probabilities of relevance \( Pr[rel|q, d] \), which are directly used for merging results together in one list, without further normalisation.

### 5.2 Resource selection in peer-to-peer networks

Similarly to \[19\], the resource selection framework is integrated into the hierarchical network in a decentralised way \[22\]: Each hub receiving the query selects those neighbour leaves and neighbour hubs which minimise overall costs, employing resource descriptions of all neighbours (including hub descriptions).

Figure 2(a) depicts the message flow in the whole network; here, thick lines denote connections over which messages are sent (the numbers indicate the order), while thin dotted lines are other connections. In this example, hub 1 is the “initial hub”, the first hub which receives the query.

Figure 2(b) shows which services are involved on the hubs and on the leaves, and how they interact: Each leaf hosts arbitrarily many leaf inventory services and attached search services. Each hub hosts three service instances:

- A resource selection service computes an optimum selection, using the techniques described above.
- A result merging service receives the results from search services and returns a single ranked list. Currently, the documents are re-ranked according to the probabilities of relevance, but a different technique could be plugged in easily.
- A (hub) search service receives the query, just like a search service backed by a DL. It then utilises other services on the hub and on neighbour peers for answering a user query.

In addition, the hub inventory service instance stores one description vector for each collection, together with further information like the number \( |DL| \) of documents in the collection or the constant \( c \).

In environments with heterogeneous schemas, schema mapping services have to be employed (see section 7 for a detailed description).

### 5.3 Experimental results

The decision-theoretic framework has proven to be effective, and to yield competitive results compared to CORI \[23\]. This section reports the quality of DTF applied to peer-to-peer networks.\(^6\)

Experiments have been conducted on a large, heterogeneous WT10g-based test-bed \[19\] with 2,500 collections (each on a leaf).

\(^6\)An extended evaluation is described in \[22\].
6. BROWSING

For browsing, the goal is to present an overview over the content available in the whole network, and to enable the user to interactively explore portions of the data in more detail. Scatter/Gather [7] is a highly interactive cluster-based browsing paradigm in which an overview of the whole dataset or of a selected subset is generated by clustering and presented to the user who can again select parts of the overview (or the whole) for further inspection, and so on. The generated overviews consist of centroids of clusterings of the selected data. In pre-processed Scatter/Gather [6], it was shown that these clusterings do not have to be computed directly from the underlying documents. It suffices to consider representatives of datasets instead, which contain the most important statistical properties of the data they represent. In [6], these representatives were cluster summaries from a pre-computed cluster hierarchy. In Pepper, we use resource descriptions of different granularities for the same purpose. Thus, our approach relies on off-line clustering for the computation of resource descriptions of different granularities, and uses online clusterings of these descriptions for generating browsable overviews of portions of the data.

6.1 Resource descriptions of different granularities

As described in section 4, each leaf peer generates a cluster hierarchy for its own documents and provides the individual cluster descriptions to other services. In addition to the term statistics, each cluster description for a cluster $C_i$ also contains a homogeneity score $h(C_i)$, measuring the mean similarity of the individual elements (resource descriptions in the cluster) $d$ to the cluster centroid $c_i$:

$$h(C_i) = \frac{1}{|C_i|} \sum_{d \in C_i} \text{sim}(d, c_i)$$

Table 1: Centralised vs. decentralised resource selection

<table>
<thead>
<tr>
<th></th>
<th>Centralised</th>
<th>Decentralised</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@5</td>
<td>0.7958 / 0.0%</td>
<td>0.7058 / -11.3%</td>
</tr>
<tr>
<td>P@10</td>
<td>0.6586 / 0.0%</td>
<td>0.5721 / -13.1%</td>
</tr>
<tr>
<td>P@15</td>
<td>0.5531 / 0.0%</td>
<td>0.4896 / -11.5%</td>
</tr>
<tr>
<td>P@20</td>
<td>0.4767 / 0.0%</td>
<td>0.4295 / -9.9%</td>
</tr>
<tr>
<td>P@30</td>
<td>0.3774 / 0.0%</td>
<td>0.3456 / -8.4%</td>
</tr>
<tr>
<td>Avg. Precision</td>
<td>0.2565 / 0.0%</td>
<td>0.2391 / -6.8%</td>
</tr>
<tr>
<td>Precision</td>
<td>0.2688 / 0.0%</td>
<td>0.2515 / -6.4%</td>
</tr>
<tr>
<td>Recall</td>
<td>0.2576 / 0.0%</td>
<td>0.2417 / -6.2%</td>
</tr>
<tr>
<td>#Hops</td>
<td>103.4 / 0.0%</td>
<td>59.9 / -42.1%</td>
</tr>
</tbody>
</table>

Table 1 compares the decentralised selection used in Pepper with a centralised selection, where the peer-to-peer network is only used for collecting all cost estimations, and a single selection is performed. While retrieval quality is rather good for decentralised selection, it is not surprisingly outperformed by the centralised variant (6-13%). However, the important advantage is its efficiency (about 40% less hops).

A comparison of our approach with other methods is difficult, as these experiments are not comparable. However, the decentralised DTF selection approach seems to be competitive to the results reported in [19]. Future work will concentrate on a direct comparison of the different approaches.

6.2 Scatter/Gather browsing in Pepper

Figure 3 shows the services involved in a browsing session and their communication. When the user starts browsing, the browsing service (running on a peer) requests the initial overview over the whole collection from its hub. Like in [17], each hub knows all other hubs in the network. To create the desired overview, the hub inventory service collects the neighbourhood resource descriptions from the other hubs, merges them with its own resource description, and applies a fast online clustering algorithm (in the current implementation, Buckshot). The result is a set of cluster descriptions which are presented to the user.

Each hub maintains a list of the resource descriptions of the leaves to which it is directly connected. To provide a browsable overview over the content in this neighbourhood, the hub further organises the leaf descriptions into a cluster hierarchy. The top layer of this hierarchy forms a content-based partitioning of the available data and is exported on request to other hubs in the form of cluster descriptions. Thus, hubs provide an overview over the content within their region of the network at any point of time.

Whenever a peer joins or leaves the network, this is reflected in the set of resource descriptions on its hub. In a large neighbourhood, and depending on the dynamics of the neighbourhood and the computing resources of the responsible hub, the cluster hierarchy is not recomputed on every join or leave event, but in fixed intervals of time. Between these clustering phases, whenever a leaf joins a neighbourhood, it is integrated into the existing cluster hierarchy of the hub by following the path with the most similar clusters down to the level of single leaf descriptions, and only the cluster descriptions on this path have to be updated.

Any hierarchical clustering algorithm can be used to create the cluster hierarchies, respectively any partitioning clustering algorithm which is applied repeatedly. In the current first prototype implementation of the browsing components, we employ Buckshot ([7], a variation of k-means) and use the cosine distance between term vectors as similarity measure $\text{sim}$. The same term weights as in the resource selection case are used, and the browsing components therefore rely on the same representation of the data, but with a different interpretation. Obviously, it would be desirable to apply a probabilistic measure instead, like in the case of resource selection, but this remains to be investigated.
The user selects one or more clusters, the focus set, for further inspection. Using the homogeneity scores \( h(C) \), this focus set is expanded by replacing the most heterogeneous clusters with their elements (which are themselves, by construction, resource descriptions of clusters), until the size of the focus set reaches the maximum number of elements which can be clustered fast enough for online browsing. This expansion step is very similar to the expansion step in [6], except that we expand the most heterogeneous clusters rather than the largest ones. The resource descriptions of the new elements have to be requested from the inventory services on either the hub or, on deeper levels of the hierarchy, another peer. As soon as the focus set has reached its maximum size, the resource descriptions in the focus set are clustered online and presented to the user for further browsing.

Apart from the hub and leaf inventory services and the browsing service, additional services may be involved in the expansion phase. In the case of heterogeneous schemas, for example, resource descriptions and documents have to be translated to the browsing schema (the schema which the user selected for the browsing session). Again, this is the task of schema mapping services (described in the next section).

The limitation on the size of the focus set guarantees that only a limited number of peers have to be contacted during the expansion phase. Cluster resource descriptions are retrieved only as far as necessary (with the IDs of their direct children instead of the complete underlying hierarchies). Furthermore, hubs can cache resource descriptions. Each hub remembers which resource descriptions have changed (due to leaf peer dynamics), and when. The requesting hub which collects resource descriptions for browsing can send a timestamp with the request, and only those descriptions which have changed since then are sent again.

Still, in comparison with the resource selection scenario, the proposed solution for browsing has a major drawback: The computation of the initial overview requires contacting all hubs in the network and then clustering their resource descriptions online. Although the resource descriptions can be pruned for this purpose to a very small number of terms, this approach is not scalable beyond a certain number of hubs. To ameliorate this problem, suppose that the content of a hub region changes slowly, even if the individual peers show more dynamics, the overview over the hub descriptions could be created regularly and offline, instead of online and for each browsing session individually. Even then, however, the hubs would still have to maintain a list of all other hubs. Possible solutions to this problem are currently under research.

### 7. SCHEMA MAPPING

As mentioned in section 4, documents are modelled as a linear list of attributes. Examples are Dublin Core or BibTeX.

Schema mapping services assist other services in heterogeneous environments. When collections employ different schemas, documents and queries have to be converted between heterogeneous schemas. This task is accomplished by three kinds of schema mapping services: query transformation services, document transformation services and resource description transformation services (see below). For the retrieval task, the query has to be transformed from the user (target) schema into the DL schema, and the retrieved documents have to be transformed back. As the resource descriptions use the DL schema, query transformation has to be performed prior to the resource selection step. For browsing, resource descriptions of clusters have to be mapped during the focus set expansion phase, and document transformation is required as soon as a browsing session reaches the leaves of the cluster hierarchy.

Each schema mapping service is specialised in mapping from one fixed schema onto another fixed schema. Multiple schema mapping services have to be chained to map between two schemas for which no direct mapping is available. Pepper employs the sPLMap framework [24], which allows for uncertain mappings. These are required when schemas of different granularity are investigated. For instance, if a schema with the general attribute “creator” has to be mapped onto a schema with two attributes “author” and “editor”, there is no precise mapping.

The sPLMap framework follows a declarative approach for human-readable schema mappings. The schema attributes are mapped onto relations in probabilistic Datalog [13], a probabilistic extension to Horn logics. Queries are logical rules. Schema mappings are defined by rules in a declarative way, e.g.:

\[
0.7 \text{author}(x, y) \leftarrow \text{creator}(d, x), \\
0.3 \text{editor}(x, y) \leftarrow \text{creator}(d, x).
\]

These rules state that a creator is an author with probability of 70%, and an editor with probability of 30%. These simple rules can be augmented if the query transformation also has to include search operators (when a specific search operator is not available in the target schema, e.g. soundex similarity for author names), or when the values have to be converted (for instance between different date formats, e.g. from “2004-31-12” to “December 31, 2004” or even across languages to “31. Dezember 2004”).

Different techniques can be used for learning schema mappings as well, e.g. LSD [9] or sPLMap [24].

### 8. CONCLUSION AND OUTLOOK

This paper describes the Pepper prototype, a peer-to-peer system for information retrieval in federated digital libraries. The flexible Pepper architecture seamlessly combines a service-oriented architecture with hierarchical peer-to-peer networks (on the basis of JXTA). On top of this architecture, several services provide two information access modes, searching and browsing. Searching and browsing employ the same statistical data about the digital libraries. Hence, a common service collects and stores these statistical descriptions of collections, and can be utilised for both access modes. This shows one of the major advantages of our approach: Services can be reused for different aspects of the search process, namely retrieval and browsing.

Another advantage of our architecture is that it is particularly easy to replace a service implementation by a new, better one as long as the service interface remains unchanged. For example, a new resource selection approach can be easily integrated by replacing that single service, keeping e.g. the result merging service and the hub search service unchanged.

Based on existing components, we are currently implementing the overall system. Part of our future work will focus on improving the single components. E.g., the quality of the resource selection approach is already good, but should be improved further. The browsing process will be further decentralised, and the application of other similarity measures and clustering algorithms will be investigated.

The two access modes could be coupled more tightly. Searching for a known document, and requesting an overview of the complete collection, are just two extremes which are distinguished by the granularity (or level of abstraction) of the desired resource(s). For instance, the system could generate a clustering of large query results, so that a user can get an overview over the document space by browsing. In addition, the cluster description hierarchies in the background could also be used to find documents similar to relevant ones.
9. ACKNOWLEDGEMENTS

Part of this work is supported in part by the DFG (project “Pepper”, grant BIB47 DOuv 02-01).

10. REFERENCES


