LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs*

Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis

Dept. of Electronic and Computer Engineering,
Technical University of Crete, GR73100 Chania, Crete, Greece
{trifon, sidraios, manolis}@intelligence.tuc.gr

Abstract. We present a digital library architecture based on distributed hash tables. We discuss the main components of this architecture and the protocols for offering information retrieval and information filtering functionality. We present an experimental evaluation of our proposals.

1 Introduction

We present a digital library (DL) architecture based on ideas from traditional distributed Information Retrieval and recent work on peer-to-peer (P2P) networks. Our architecture, called LibraRing (from the words *libra*ry and *ring*), is hierarchical like the ones in [12,9] but uses a *distributed hash table* (DHT) to achieve robustness, fault-tolerance and scalability in its routing and meta-data management layer. DHTs are the second generation *structured* P2P overlay networks devised as a remedy for the known limitations of earlier P2P networks such as Napster and Gnutella [15].

There are two kinds of basic functionality that we expect this DL architecture to offer: *information retrieval (IR)* and *publish/subscribe (pub/sub)*. In an IR scenario, a user can pose a *query* (e.g., "I am interested in papers on bio-informatics") and the system returns information about matching resources. In a pub/sub scenario (also known as *information filtering (IF)* or *selective dissemination of information (SDI)*), a user posts a *subscription* (or *profile* or *continuous query*) to the system to receive notifications whenever certain events of interest take place (e.g., when a paper on bio-informatics becomes available).

We define the main components of our architecture: *super-peers, clients* and *providers.* Providers are used to expose the content of information sources to the network, while clients are used by information consumers. Super-peers form an overlay network that offers a robust, fault-tolerant and scalable means for routing messages and managing resource meta-data and queries. The main architectural contribution of our work is the extension of the DHT Chord protocols [15] with IR and pub/sub functionality in the context of a super-peer network.

Publications and subscriptions in our architecture could be expressed using any appropriate language (e.g., XML and XPath). Whatever language is chosen

* This work was supported in part by project Evergrow. Christos Tryfonopoulos is partially supported by a Ph.D. fellowship from the program Heraclitus of the Greek Ministry of Education.

A. Rauber et al. (Eds.): ECDL 2005, LNCS 3652, pp. 25–36, 2005.

will have a serious effect on the DHT protocols because the DHT is the layer in which publications and subscriptions live (are indexed). In the rest of this paper, we assume that publications and subscriptions will be expressed using a well-understood attribute-value model, called $\mathcal{AWPS}$ in [10]. $\mathcal{AWPS}$ is based on *named attributes* with value *free text* interpreted under the Boolean and vector models VSM or LSI. The query language of $\mathcal{AWPS}$ allows Boolean combinations of comparisons A *op* v, where A is an attribute, v is a text value and *op* is one of the operators "equals", "contains" or "similar" ("equals" and "contains" are Boolean operators and "similar" is interpreted using the VSM or LSI model).

The research presented in this paper is a continuation of our previous work on the DL information alert architecture DIAS [10] and the system P2P-DIET [7]. The main difference of the current paper from [10,7] is the definition of an architecture for DLs, brand new protocols that are extensions of DHTs and performance results that illustrate the strengths and weaknesses of our approach.

The rest of this paper is as follows. Section 2 positions our paper with respect to related work in the areas of IR, IF and P2P systems. Section 3 introduces the model $\mathcal{AWPS}$ while Section 4 presents the Chord DHT. Section 5 discusses the proposed DL architecture and an application scenario. Section 7 presents the LibraRing protocols, while Section 8 summarizes our experimental evaluation. Finally, Section 9 concludes the paper.

2 Related Work

The problem of IR and IF in DL architectures that utilize P2P networks has recently received considerable attention. Here we only discuss papers that we judge most relevant to our work (because they combine an emphasis on DLs, models and languages based on IR, and techniques from P2P networks).

The hierarchical 2-tier architecture of LibraRing is similar to the ones of file sharing systems currently deployed on the Internet, e.g., Kazaa, Gnutella2 and Fast Track, and has also been studied by P2P researchers [2,13]. From these proposals, only Edutella [13,4] uses a structured overlay network for its routing layer; all other approaches rely on techniques from unstructured P2P networks. Some recent proposals targeting DLs also rely on a similar architecture [12,11,7,4,9]. Papers coming from the IR community prefer to use the term *hub node* or *directory node* instead of super-peer or ultra-peer and the term *leaf node* instead of client [12,11,9]. However in our case super-peers have more responsibilities that just to provide directory services which is the case for hubs.

In [11] the problem of content-based retrieval in distributed DLs focusing on resource selection and document retrieval is studied. A 2-tier hierarchical P2P network is proposed, where DLs (represented by leaf nodes) cluster around directory nodes that form an unstructured P2P network in the 2nd level of the hierarchy. In a related paper [12], the authors define the concept of neighborhood in hierarchical P2P networks and use this concept to devise a method for hub selection and ranking. The PlanetP [5] system uses an unstructured P2P network where nodes propagate Bloom filter summaries of their indices to the network using a gossiping algorithm. Each peer uses a variation of *tf/idf* to decide what

nodes to contact to answer a query. In pSearch [19] the authors propose to use the CAN DHT protocol [14] and semantic document vectors (computed using LSI) to efficiently distribute document indices in a P2P network. PIRS [24] uses an unstructured P2P network and careful propagation of metadata information to be able to answer queries in highly dynamic environments. Finally, OverCite [17] is a recent proposal to build a distributed version of CiterSeer using DHTs.

Early work on IF includes SIFT [22,23] which uses the Boolean and VSM models, and InRoute [3] which is based on inference networks. Both of these systems are centralised although some issues related to distribution have been studied in SIFT [23]. Recently, a new generation of IF systems has tried to address the limitations imposed by centralized approaches by relying on ideas from P2P networks. The system P2P-DIET [7], that builds on the earlier proposal DIAS [10], is an IR and IF system that uses the model $\mathcal{AWPS}$ and is implemented as an unstructured P2P network with routing techniques based on shortest paths and minimum-weight spanning trees. pFilter [18] uses a hierarchical extension of CAN to filter unstructured documents and relies on multi-cast trees to notify subscribers. Architectural considerations regarding the development of an IF system for DLs were first studied by Hermes [6].

None of the papers cited above provides a comprehensive architecture and protocols for the support of *both* IR and IF functionality in DLs using DHTs. This is the emphasis of our work which is presented in the forthcoming sections.

3 The Data Model $\mathcal{AWPS}$

We will use a well-understood attribute-value model, called $\mathcal{AWPS}$ in [10]. A *(resource) publication* is a set of attribute-value pairs (A, s), where A is a *named attribute*, s is a *text* value and all attributes are *distinct*. The following is an example of a publication:

$$\{\ (AUTHOR, \text{“}John\ Smith\text{”}), (TITLE, \text{“}Information\ dissemination\ in\ P2P\ ...\text{”}),\\ (ABSTRACT, \text{“}In\ this\ paper\ we\ show\ that\ ...\text{”})\ \}$$

The query language of $\mathcal{AWPS}$ offers *equality*, *containment* and *similarity* operators on attribute values. The containment operator is interpreted under the Boolean model and allows Boolean and *word-proximity* queries. The similarity operator is defined as the cosine of the angle of two vectors corresponding to text values from a publication and a query. Vector representations of text values can be computed as usual using the VSM or LSI models (but only the VSM model has been used in our implementation and experiments).

Formally, a *query* is a conjunction of atomic queries of the form $A = s$, $A \sqsupseteq wp$ or $A \sim_k s$, where A is an attribute, s is a text value, wp is a conjunction of words and proximity formulas with only words as subformulas, and k is a *similarity threshold* i.e., a real number in the interval $[0, 1]$. Thus, queries can have two parts: a part interpreted under the Boolean model and a part interpreted under the VSM or LSI model. The following is an example of a query:

$$(AUTHOR = \text{“}John\ Smith\text{”})\ \wedge\ (TITLE \sqsupseteq P2P \wedge (information \prec_{[0,0]} alert)) \wedge\\ (ABSTRACT \sim_{0.7} \text{“}P2P\ architectures\ have\ been...\text{”})$$

This query requests resources that have *John Smith* as their author, and their title contains the word *P2P* and a word pattern where the word *information* is immediately followed by the word *alert*. Additionally, the resources should have an abstract similar to the text value *"P2P architectures have been ..."* with similarity greater than 0.7.

4 Distributed Hash Tables

We use an extension of the Chord DHT [15] to implement our super-peer network. Chord uses a variation of *consistent hashing* [8] to map keys to nodes. In the consistent hashing scheme each node and data item is assigned a m-bit identifier where m should be large enough to avoid the possibility of different items hashing to the same identifier (a cryptographic hashing function such as SHA-1 is used). The identifier of a node can be computed by hashing its IP address. For data items, we first have to decide a key and then hash this key to obtain an identifier. For example, in a file-sharing application the name of the file can be the key (this is an application-specific decision). Identifiers are ordered in an *identifier circle (ring)* modulo 2^m i.e., from 0 to $2^m - 1$. Figure 1(b) shows an example of an identifier circle with 64 identifiers ($m = 6$) and 10 nodes.

Keys are mapped to nodes in the identifier circle as follows. Let H be the consistent hash function used. Key k is assigned to the first node which is equal or follows $H(k)$ clockwise in the identifier space. This node is called the *successor* node of identifier $H(k)$ and is denoted by $successor(H(k))$. We will often say that this node is *responsible* for key k. For example in the network shown in Figure 1(b), a key with identifier 30 would be stored at node $N32$. In fact node $N32$ would be responsible for all keys with identifiers in the interval $(21, 32]$.

If each node knows its successor, a query for locating the node responsible for a key k can always be answered in $O(N)$ steps where N is the number of nodes in the network. To improve this bound, Chord maintains at each node a routing table, called the *finger table*, with at most m entries. Each entry i in the finger table of node n, points to the first node s on the identifier circle that succeeds identifier $H(n) + 2^{i-1}$. These nodes (i.e., $successor(H(n) + 2^{i-1})$ for $1 \leq i \leq m$) are called the *fingers* of node n. Since fingers point at repeatedly doubling distances away from n, they can speed-up search for locating the node responsible for a key k. If the finger tables have size $O(\log N)$, then finding a successor of a node n can be done in $O(\log N)$ steps with high probability [15].

To simplify joins and leaves, each node n maintains a pointer to its *predecessor* node i.e., the first node *counter-clockwise* in the identifier circle starting from n. When a node n wants to join a Chord network, it finds a node n' that is already in the network using some out-of-band means, and then asks n' to find its position in the network by discovering n's successor [16]. Every node n runs a *stabilization* algorithm periodically to find nodes that have recently joined the network by asking its successor for the successor's predecessor p. If p has recently joined the network then it might end-up becoming n's successor. Each node n periodically runs two additional algorithms to check that its finger table and predecessor pointer is correct [16]. Stabilization operations may affect queries

by rendering them slower or even incorrect. Assuming that successor pointers are correct and the time it takes to correct finger tables is less than the time it takes for the network to double in size, queries can still be answered correctly in $O(\log N)$ steps with high probability [16].

To deal with node failures and increase robustness, each Chord node n maintains a *successor list* of size r which contains n's first r successors. This list is used when the successor of n has failed. In practice even small values of r are enough to achieve robustness [16]. If a node chooses to leave a Chord network voluntarily then it can inform its successor and predecessor so they can modify their pointers and, additionally, it can transfer its keys to its successor. Any node joining or leaving a Chord network can use $O(\log^2 N)$ messages to make all successor pointers, predecessor pointers and finger tables correct [15].

5 The LibraRing Architecture

A high-level view of the LibraRing architecture is shown in Figure 1(a). Nodes can implement any of the following types of services: *super-peer service, provider service* and *client service*.

Super-peer service. Nodes implementing the super-peer service (*super-peers*) form the message routing layer of the network. Each super-peer is responsible for serving a fraction of the clients by storing continuous queries and resource publications, answering one-time queries, and creating notifications. The super-peers run a DHT protocol which is an extension of Chord. The role of the DHT in LibraRing is very important. First of all, it acts as a rendezvous point for information producers (providers) and information consumers (clients). Secondly, it serves as a robust, fault-tolerant and scalable routing infrastructure. When the number of super-peers is small, each node can easily locate others in a single hop by maintaining a full routing table. When the super-peer network grows in size, the DHT provides a scalable means of locating other nodes as we discussed in Section 4. Finally, by serving as a global metadata index that is partitioned among super-peers, the DHT facilitates building a distributed metadata repository that can be queried efficiently.

Client service. Nodes implementing the client service are called *clients*. A client connects to the network through a single super-peer node, which is its *access point*. Clients can connect, disconnect or even leave the system silently at any time. Clients are information consumers: they can pose one-time queries and receive answers, subscribe to resource publications and receive notifications about published resources that match their interests. If clients are not on-line, notifications matching their interests are stored by their access points and delivered once clients reconnect. Resource requests are handled directly by the client that is the owner of the resource.

Provider service. This service is implemented by information sources that want to expose their contents to the clients of LibraRing. A node implementing the provider service (*provider*) connects to the network through a super-peer

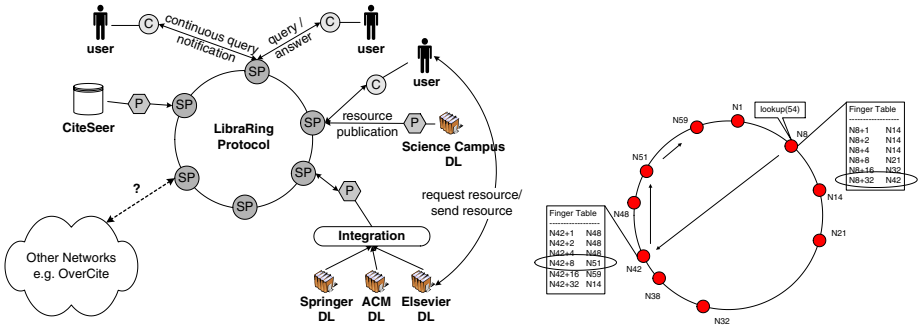


Fig. 1. The architecture of LibraRing and an example of a lookup operation over a Chord ring with m=6

which is its access point. To be able to implement this service, an information source should creates meta-data for the resources it stores using data model $\mathcal{AWPS}$, and publish it to the rest of the network using its access point.

As an example of an application scenario let us consider a university with 3 geographically distributed campuses (Arts, Sciences and Medicine) and a local digital library in each campus (see Figure 1(a)). Each campus maintains its own super-peer, which provides an access point for the provider representing the campus digital library, and the clients deployed by users. A university might be interested in making available to its students and staff, in a timely way, the content provided by other publishers (e.g., CiteSeer, ACM, Springer, Elsevier). Figure 1(a) shows how our architecture can be used to fulfill this requirement. An integration layer is used to unify different types of DLs. At this level, we also expect to see *observer modules* (as in [6]) for information sources that do not provide their own alerting service. This modules will query the sources for new material in a scheduled manner and inform providers accordingly.

Questions of interoperability are not discussed in this paper. Figure 1(a) assumes that some of them are solved at the level of providers with well-known integration techniques, and leaves open the question of how to interoperate with other P2P networks e.g., the DHT implementing OverCite [17]. The latter question is currently under study in project Evergrow[1] that is funding this work.

6 Extensions to the Chord API

To facilitate message sending between nodes we will use the function `send(`msg, I`)` to send message msg from some node to node $successor(I)$, where I is a node identifier. Function `send()` is similar to Chord function `lookup(`I`)` [15], and costs $O(logN)$ overlay hops for a network of N nodes. When function `send(`msg, I`)` is invoked by node S, it works as follows. S contacts S', where $id(S')$ is the greatest identifier contained in the finger table of S, for which $id(S') \leq I$ holds.

[1] `http://www.evergrow.org/`

Upon reception of a `send()` message by a node S, I is compared with $id(S)$. If $id(S) < I$, then S just forwards the message by calling `send(`msg, I`)` itself. If $id(S) \geq I$, then S processes msg since it is the *intended recipient*.

Our protocols described in Section 7 also require that a node is capable of sending the *same* message to a group of nodes. This group is created dynamically each time a resource publication or a query submission takes place, so multicast techniques for DHTs such as [1] are not applicable. The obvious way to handle this over Chord is to create k different `send()` messages, where k is the number of different nodes to be contacted, and then locate the recipients of the message in an *iterative* fashion using $O(k \log N)$ messages. We have implemented this algorithm for comparison purposes.

We have also designed and implemented function `multiSend(`msg, L`)`, where L is a list of k identifiers, that can be used to send message msg to the k elements of L in a *recursive* way. When function `multiSend()` is invoked by node S, it works as follows. Initially S sorts the identifiers in L in ascending order clockwise starting from $id(S)$. Subsequently S contacts S', where $id(S')$ is the greatest identifier contained in the finger table of S, for which $id(S') \leq head(L)$ holds, where $head(L)$ is the first element of L. Upon reception of a `multiSend()` message, by a node S, $head(L)$ is compared with $id(S)$. If $id(S) < head(L)$, then S just forwards msg by calling `multiSend()` again. If $id(S) \geq head(L)$, then S processes msg since this means that it is *one* of the intended recipients contained in list L (in other words, S is responsible for key $head(L)$). Then S creates a new list L' from L in the following way. S deletes all elements of L that are smaller or equal to $id(S)$, starting from $head(L)$, since S is responsible for them. In the new list L' that results from these deletions, we have that $id(S) < head(L')$. Finally, S forwards msg to node with identifier $head(L')$ by calling `multiSend(`msg, L'`)`. This procedure continues until all identifiers are deleted form L. The recursive approach has in practice a significantly better performance than the iterative method as we show in Section 8.

7 The LibraRing Protocols

In this section we describe in detail the way clients, providers and super-peers join and leave the network. We also describe resource publication and query submission protocols. We use functions $key(n)$, $ip(n)$ and $id(n)$ to denote the key, the IP address and the identifier of node n respectively.

7.1 Client Join

The *first time* that a client C wants to connect to the LibraRing network, it has to follow the join protocol. C must find the IP address of a super-peer S using out-of-band means (e.g., via a secure web site that contains IPs for the super-peers that are currently online). C sends to S message NewClient$(key(C), ip(C))$ and S adds C in its *clients table* (CT), which is a hash table used for identifying the peers that use S as their *access point*. $key(C)$ is used to index clients in CT, while each CT slot stores contact information about the client, its status

(connected/disconnected) and its stored notifications (see Section 7.6). Additionally, S sends to C an acknowledgement message AckNewClient($id(S)$). Once C has joined, it can use the connect/disconnect protocol (to be described below) to connect and disconnect from the network.

Providers use a similar protocol to join a LibraRing network.

7.2 Client Connect/Disconnect

When a client C wants to connect to the network, it sends to its access point S message ConnectClient($key(C), ip(C), id(S)$). If $key(C)$ exists in the CT of S, C is marked as connected and stored notifications are forwarded to it. If $key(C)$ does not exist in CT, this means that S was not the access point of C the last time that C connected (Section 7.7 discusses this case).

When a client C wants to disconnect, it sends to its access point S message DisconnectClient($key(C), ip(C)$). S marks C as disconnected in its CT without removing information related to C, since this information will be used to create stored notifications for C while C is not online (see Section 7.6).

Providers connect to and disconnect from the network in a similar way.

7.3 Resource Indexing

A resource is indexed in three steps. In the first step a provider P constructs a publication $p = \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\}$ (the resource description) and sends message PubResource($key(P), ip(P), key(p), p$) to its access point S.

In step two, S forwards p to the appropriate super-peers as follows. Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. Then p is sent to *all* nodes with identifiers in the list $L = \{H(w_j) : \ w_j \in D_1 \cup \cdots \cup D_n\}$. The subscription protocol guarantees that L is a superset of the set of identifiers responsible for queries that match p. Subsequently S removes duplicates and sorts L in ascending order clockwise starting from $id(S)$. This way we obtain less identifiers than the distinct words in $D_1 \cup \cdots \cup D_n$, since a super-peer may be responsible for more than one words contained in the document. Having obtained L, S indexes p by creating message msg =IndexResource($ip(P), key(P), ip(S), key(p), p$), and calling function `multiSend(`msg, L`)`.

Finally in the third step, each super-peer S' that receives this message stores p in an *inverted index* that will facilitate matching against one-time queries that will arrive later on at S'.

7.4 Submitting an One-Time Query

In this section we show how to answer one-time queries containing Boolean and vector space parts (denoted as queries of type T1) or only vector space parts (denoted as queries of type T2). The first type of queries is always indexed under its Boolean part. Let us assume that a client C wants to submit a query q (of type T1) of the form $\bigwedge_{i=1}^{m} A_i = s_i \ \wedge \ \bigwedge_{i=m+1}^{n} A_i \sqsupseteq wp_i \ \wedge \ \bigwedge_{i=n+1}^{k} A_i \sim_{a_i} s_i$.

The following three steps take place. In step one, C sends to its access point S message SubmitQ($key(C), ip(C), key(q), q$).

In the second step, S randomly selects a single word w contained in any of the text values $s_1, \ldots, s_m$ or word patterns $wp_{m+1}, \ldots, wp_n$ and computes $H(w)$ to obtain the identifier of the super-peer storing publications that can match q. Then it sends message msg =PoseQuery$(ip(C), key(C), ip(S), key(q), q)$ by calling function `send(`$msg, H(w)$`)`.

If q is of the form $A_{n+1} \sim_{a_1} s_1 \wedge \ldots \wedge A_n \sim_{a_n} s_n$ (query type T2) then step two is modified as follows. Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. q has to be sent to *all* super-peers with identifiers in the list $L = \{H(w_j) : w_j \in D_1 \cup \cdots \cup D_n$. To do so, S removes duplicates, sorts L in ascending order clockwise starting from $id(S)$ and sends message msg =PoseQuery$(ip(C), key(C), ip(S), key(q), q)$ by calling `multiSend(msg,L)`.

In step three, each super-peer that receives an one-time query q, it matches it against its local publication store to find out which providers have published documents that match q and delivers answers as discussed in Section 7.6.

7.5 Pub/Sub Functionality

This section describes how to extend the protocols of Sections 7.4 and 7.3 to provide pub/sub functionality. To index a continuous query *cq* the one-time query submission protocol needs to be *modified*. The first two steps are identical, while the third step is as follows. Each super-peer that receives *cq*, it stores *cq* in its local continuous query data structures to match it against incoming publications. A super-peer S uses a hash table to index all the atomic queries of *cq*, using as key the attributes $A_1, \ldots, A_k$. To index each atomic query, three different data structures are also used: (i) a hash table for text values $s_1, \ldots, s_m$, (ii) a trie-like structure that exploits common words in word patterns $wp_{m+1}, \ldots, wp_n$, and (iii) an inverted index for the most "significant" words in text values $s_{n+1}, \ldots, s_k$. S' utilises these data structures at filtering time to find quickly all continuous queries *cq* that match an incoming publication p. This is done using an algorithm that combines algorithms BestFitTrie [21] and SQI [23].

To index a resource, the protocol of Section 7.3 needs to be *extended*. The first two steps are identical, while in the third step, each super-peer that receives p matches it against its local continuous query database using the algorithms BestFitTrie and SQI.

7.6 Notification Delivery

Assume a super-peer S that has to deliver a notification n for a continuous query *cq* to client C. It creates message msg =Notification$(ip(P), key(P), pid(p), qid(cq))$, where P is the provider that published the matching resource and sends it to C. If C is not online, then S sends msg to S', where S' is the access point of C, using $ip(S')$ associated with *cq*. S' stores msg, to deliver it to C upon reconnection. If S' is also off-line msg is sent to the $successor(id(S'))$, by calling function `send(`$msg, successor(id(S'))$`)`. Answers to one-time queries are handled in a similar way. In case that more that one answers or notifications have to be delivered, function `multiSend()` is used.

7.7 Super-Peer Join/Leave

To join LibraRing network, a super-peer S must find the IP address of another super-peer S' using out-of-band means. S creates message NewSPeer($id(S)$, $ip(S)$) and sends it to S' which performs a lookup operation by calling `lookup (id(S))` to find $S_{succ} = successor(id(S))$. S' sends message AckNewSPeer $(id(S_{succ}), ip(S_{succ}))$ to S and S updates its successor to S_{succ}. S also contacts S_{succ} asking its predecessor and the data that should now be stored at S. S_{succ} updates its predecessor to S, and answers back with the contact information of its previous predecessor, S_{pred}, and all continuous queries and publications that were indexed under key k, with $id(S) \leq k < id(S_{pred})$. S makes S_{pred} its predecessor and populates its index structures with the new data that arrived. After that S populates its finger table entries by repeatedly performing lookup operations on the desired keys.

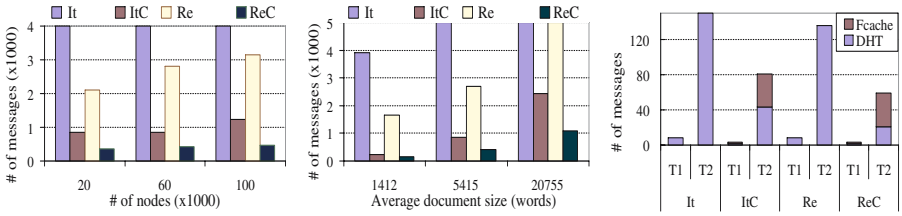
When a super-peer S wants to leave LibraRing network, it constructs message DisconnectSPeer($id(S), ip(S), id(S_{pred}), ip(S_{pred}), data$), where *data* are all the continuous queries, published resources and stored notifications of off-line peers that S was responsible for. Subsequently, S sends the message to its successor S_{succ} and notifies S_{pred} that its successor is now S_{succ}. Clients that used S as their access point connect to the network through another super-peer S'. Stored notifications can be retrieved through $successor(id(S))$.

8 Experimental Evaluation

In [20] we present a detailed evaluation of DHTrie, a set of protocols that are essentially the LibraRing protocols in the case that all nodes in the system are equal DHT nodes (i.e., there are no super-peers). [20] deals only with the pub/sub case and evaluates the DHTrie protocols only under queries of type T1. In this section we continue this evaluation for the complete LibraRing protocols using the same data and query set. As in [21,20], we use a corpus of 10426 documents downloaded from CiteSeer and synthetically generated queries. In all graphs the y-axes has been truncated to show clearly the best performing algorithms.

We have implemented and experimented with four variations of the LibraRing protocols. The first one, named *It*, utilises the iterative method in the publication protocol. The second algorithm, named *ItC*, utilises the iterative method and an additional routing table called FCache. *FCache* is a routing table that stores the IP addresses of super-peers responsible for *frequent words* and is exploited for reaching nodes in a single hop. A detailed description and performance evaluation for FCache is given in [20]. The third algorithm, named *Re*, utilises the recursive method in the publication protocol. Finally, *ReC* uses the recursive method and FCache and outperforms the rest of the algorithms.

Figure 2(a) shows the number of messages needed by each algorithm to index a resource publication to the appropriate super-peers for different super-peer network sizes. The algorithms remain relatively unaffected by network size since a publication needs 5% more messages for a 5 times larger network. All algorithms



(a) Varying network size (b) Varying publication size (c) Varying query type

Fig. 2. Average number of messages to index a publication (a,b) and a query (c)

show a similar increase rate, with *ReC* presenting the best performance, needing 400 messages to index an incoming publication to a network of 100K super-peers.

Publication size is an important parameter in the performance of our algorithms. Figure 2(b) shows that for small publications the use of the recursive method (contrary to FCache) does not improve performance, since algorithms *ItC* and *ReC* perform similarly. For large publications though, the use of the recursive method and FCache is shown to improve performance significantly.Additionally the increase in message cost is linear to the publication size with algorithm *ReC* exhibiting the smallest increase rate, thus showing the smallest sensitivity to publication size.

Finally Figure 2(c) shows the cost indexing a query to the network. Notice that queries of type T2 are much more expensive to index needing about 60 messages to reach the responsible super-peers, while queries of type T1 can be indexed in an average of 8 messages. Again algorithm *ReC* shows a significant improvement in message cost.

9 Outlook

We are currently implementing LibraRing in the context of project Evergrow. We are also moving to more expressive languages that combine hierarchical structure and textual information (e.g., XQuery with full-text operators).

References

1. L.O. Alima, A. Ghodsi, P. Brand, and S. Haridi. Multicast in DKS(N, k, f) Overlay Networks. In *Proc. of OPODIS*, 2003.
2. B. Yang and H. Carcia-Molina. Designing a super-peer network. In *Proceedings ICDE*, 2003.
3. J.P. Callan. Document Filtering With Inference Networks. In *Proc. of ACM SIGIR*, 1996.
4. P. A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/Subscribe for RDF-based P2P Networks. In *Proc. of ESWS*, 2004.
5. F.M. Cuenca-Acuna and T.D. Nguyen. Text-Based Content Search and Retrieval in Ad-hoc P2P Communities. In *Networking 2002 Workshops*, 2002.

6. D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – A Notification Service for Digital Libraries. In *Proc. of JCDL*, 2001.
7. S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2P-DIET: An Extensible P2P Service that Unifies Ad-hoc and Continuous Querying in Super-Peer Networks. In *Proc. of SIGMOD*, 2004. Demo paper.
8. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of STOC*, 1997.
9. I.A. Klampanos and J.M. Jose. An Architecture for Information Retrieval over Semi-Collaborating Peer-to-Peer Networks. In *Proc. of SAC*, 2004.
10. M. Koubarakis, T. Koutris, P. Raftopoulou, and C. Tryfonopoulos. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proc. of ECDL*, 2002.
11. J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proc. of CIKM*, 2003.
12. J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *Proc. of ECIR*, 2005. (to appear).
13. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proc. of WWW*, 2002.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.
15. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, 2001.
16. I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.
17. J. Stribling, I.G. Councill, J. Li, M.F. Kaashoek, D.R. Karger, R. Morris, and S. Shenker. OverCite: A Cooperative Digital Research Library. In *Proc. of IPTPS*, 2005.
18. C. Tang and Z. Xu. pFilter: Global Information Filtering and Dissemination Using Structured Overlays. In *Proc. of FTDCS*, 2003.
19. C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In *Proc. of HotNets-I '02*.
20. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks. In *Proc. of ACM SIGIR*, 2005.
21. C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proc. of ACM SIGIR*, 2004.
22. T.W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM TODS*, 19(2):332–364, 1994.
23. T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM TODS*, 24(4):529–565, 1999.
24. W.G. Yee and O. Frieder. The Design of PIRS, a Peer-to-Peer Information Retrieval System. In *Proc. of DBISP2P*, 2004.