Implementation of a framework for a DHT-based Distributed Location Service

Simone Cirani, Luca Veltri
Dpt. Information Engineering, University of Parma
Parma, Italy
E-mail: simone.cirani@gmail.com, luca.veltri@unipr.it

Abstract: **Distributed Hash Tables (DHTs) are structured peer-to-peer systems in which a number of peer nodes collectively cooperate to provide a key/value pair information storage and retrieval service. All DHTs are characterized by desirable features such as scalability, fault tolerance, and self-organization. Each DHT node is responsible for maintaining a subset of the stored information, which depends on the specific DHT algorithm. Many Internet-based applications strongly rely on a Location Service (LS), such as DNS, in order to map a URI to one or more IP addresses (and port numbers) that specify where the resource identified by the URI can actually be accessed. However, Location Services typically introduce centralization points into the architectures they are used in and therefore expose the overall system to possible failures. Because of their nature, DHT appear to be a perfect mean for setting up a Distributed Location Service (DLS). In this paper we present a Java-based framework that can be used for building a DLS independent from the specific DHT algorithm and communication protocol adopted.**

1. INTRODUCTION

Distributed Hash Tables (DHTs) are structured peer-to-peer (P2P) systems that provide an information storage and retrieval service of key/value pairs among a number of nodes. DHTs feature desirable properties such as scalability, self-organization, robustness, and fault-tolerance. Several DHT algorithms, such as Chord [1] and Kademlia [2], have been defined and successfully implemented.

DHTs rely on the cooperation of a number of nodes (peers) which collectively provide the information storage and retrieval service. Nodes are arranged on an overlay network, which is built upon an existing network, whose topology depends on the particular DHT algorithm. For instance, Chord organizes nodes on a circle, while Kademlia as leaves of a binary tree. The structure of the DHT topology affects message routing inside the DHT. Each DHT algorithm typically defines also a protocol (usually a set of RPCs) to be used for the communication and cooperation among the DHT nodes.

Applications interact with the DHT through two basic Remote Procedure Calls (RPCs), which can be abstractly defined as:

- put(key, value): this method is used to store a key/value pair into the DHT;
- get(key): this method is used to retrieve the information stored in the DHT that is associated with the given key.

The key of a resource is in general the hash of the resource name through some hashing function defined by the DHT algorithm, while the value is a short data associated to such resource (some metadata and/or the contact address where the resource can be found).

Based on such storage and retrieval service, we build a general P2P Distributed Location Servce (DLS) system capable of providing a lookup service for the binding between an URI [4], identifying a generic resource, and one or more mapped contact URIs, identifying the place where or through which the resource can be accessed. Resources could be a web service, a file, an application user agent, a user, a processor resource, or any other addressable resources. Together with each contact URI some other information useful for the LS should be stored like the expiration time, an access priority value, and, optionally, a displayable text (for example a description of the contact or a readable name). The proposed P2P LS actually maintain the mappings between resource URI contact URIs in a distributed (P2P-oriented) and reliable manner. Note that RFC 2397 [6] defines a method for mapping any (short) data within a standard URI. Using such mechanism (RFC 2397), our LS system may also be seen as a generic system for storing any kind of short data in a distributed P2P manner, providing a sort of distributed database.

In the rest of this paper we first present the layered architecture of the proposed DLS (section 2), then we describe the software implementation (section 3), and finally we present some practical developed systems based on such DLS framework (section 4), followed by some conclusions.

2. DLS ARCHITECTURE

Many applications require some lookup service to retrieve location information of some requested resources. In the simplest case, this information may be the mapping between a fully qualified domain name (FQDN), like the current DNS system, or a more sophisticated mapping that may take into account dynamic resolution and/or application level parameters. Although the DNS system may seem a basic solution for such service it suffers reliability, dynamicity, and

This work has been partially supported by the Italian Ministry for University and Research (MIUR) within the project PROFILES under the PRIN 2006 research program.

flexibility problems. For this reason, in several application environments a complete P2P-based DLS may be preferred. Examples of such environments are: i) VoIP (or general real-time communication) applications in which users may dynamically join the system through one ore more User Agents (UAs), ii) file sharing platforms, as already implemented in current P2P systems (BitTorrent, eMule, Gnutella, etc.), iii) resource sharing systems in which users share their processing and/or storing capability, iii) web services, and other dynamic remote services.

Although any single application may implement such LS in a proper and ad-hoc manner, a general and common approach may be preferred in order to let different applications to i) reuse existing protocols and corresponding implementation codes, ii) share the DLS P2P platform in order to increment the resulting availability and robustness.

Hence we propose to use a general DLS system that basically performs storing and retrieval service on a distributed table that simply maps keys to values. The keys are hash-derived by the resource URIs while the values associated to each resource are a list contact URIs with some other information useful for location service such as a text resource description or display name, the expiration date, and a priority value used when more contact URIs are provided for the same resource.

Such LS can be abstractally represented through a lookup table as shown in Figure 1.

It easy to see that this LS may be applied to a vast set of applications.

key	Values
resource-URI-1	contact-URI-1,dispaly-name-1,priority=1.0,expires=T1 contact-URI-2,dispaly-name-2,priority=1.0,expires=T2 contact-URI-3,dispaly-name-3,priority=1.0,expires=T3
resource-URI-2	contact-URI-4,dispaly-name-4,priority=1.0,expires=T4
resource-URI-3	contact-URI-5,dispaly-name-5,priority=1.0,expires=T5 contact-URI-6,dispaly-name-6,priority=1.0,expires=T6
...	...

Figure 1 – DLS abstract lookup table.

Such LS could be accessed through two very simple API calls:

- put(key, value)
- get(key)

where *key* is a resource URI, while value is a single or a set of tuples of display name, contact URI, expire time, and priority value. The *get()* method should return the set of the corresponding values (actually the contact information) associated with the given resource.

In a network-based application, such distributed LS could be implemented within a proper LS layer as represented in Figure 2.

The DLS protocol includes all mechanisms and functions to access the rest of the DLS system implemented on the other nodes according to the proper P2P system.

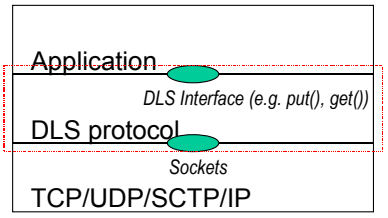

Figure 2 – General LS client layered architecture.

Considering a DHT-based P2P DLS infrastructure in which each node (actually a peer) cooperates to the maintenance of the DHT and the resulting DLS, the previous architecture can be particularized as shown in Figure 3. In such architecture the DLS protocol is composed by three sub-layers: the DLS layer, the Peer layer, and the RPC protocol used to interact with the other peers according to the selected DHT algorithm.

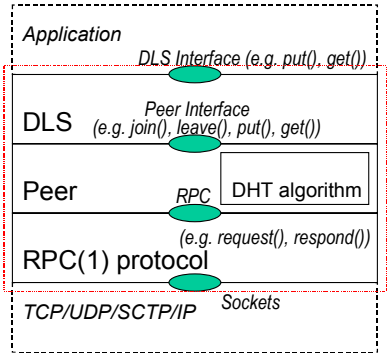

Figure 3 –DLS peer (DHT-aware) layered architecture.

Let us separately consider the various components of the proposed architecture.

2.1. DLS layer

The DLS layer provides the basic LS service to the application layer. It mainly maps the *get()* and *put()* LS methods upon the corresponding methods provided by the Peer layer that actually implements the specified DHT algorithm (Kademlia, Chord, Pastry, etc.).

All P2P specific functions (such as *join* or *leave* methods, and peer identification) are trasparent for the application layer and are masquerated by the DLS layer.

2.2. Peer layer

The Peer layer has the task to dinamically setup and maintain the DHT infrastructure, interacting with the other corresponding peers according to the chosen DHT algorithm.

It completly masquerades to the DLS layer all details about the the adopted DHT algorithm offering a trasparent and uniform interface. As result, it offers to the upper layer (the DLS) only basic peer operations common to all various DHT algorithms that are:

- *join*: this operation allows a peer to join an overlay;

- *leave*: this operation allows a peer to leave the overlay it is currently enrolled in;
- *put*: this operation allows to store a key/value pair in the DHT;
- *get*: this operation allows to retrieve the information associated with the given key.

On the other side, the actual peer remote calls depends on the chosen DHT algorithm and are mapped on the underlying RPC protocol (in Figure 3 indicated with RPC(1)).

2.3. DHT algorithm

The DHT algorithm is the actual logic implemented by the peer and used to store and retrive dynamic mapping between keys and values in a distributed fashion. At this level, the keys are the hash of the resource URI, while the mapped values are a set of tuples containing the resource contact information (contact URI, display name, expire time, and priority value).

Note that according to RFC 2397 [6], which defines a method for mapping any (short) data within a standard URI, the contact URI information may be used to encapuslate short data in place of or in addition to the actual resource contact URI. This in turn allows the DHT and the corresponding DLS to be used as generic system for storing any kind of short data in a distributed P2P manner, providing a sort of distributed database.

2.4. RPC protocol

The DHT-based P2P system requires that all peers enrolled in the DHT overlay network exchange information for the DHT setup, update, and maintainance.

The interaction between peer occurs through a request/response model according to the specific DHT algorithm implemented by the Peer layer and is mapped over the actual communication protocol provided by the RPC protocol (RPC(1) in Figure 3).

Hence this layer is responsible for trasforming the peer remote DHT methods/calls (such as join(), leave(), get(), put()) to proper request/response communication messages. In turn, the RPC protocol may use an underlying transport protocol such as TCP, UDP, SCTP, TLS or DTLS, depending on the type of the used RPC protocol (reliable/unreliable, message/stream oriented, etc.) and on the target security level.

On the receiver side this layer is responsible for receiving messages from other peers, parsing them, and calling the corresponding RPC API methods.

2.5. DHT-unaware clients and peer adapters

According to above, a DHT-aware peer cooperates to maintain the DHT and the DLS system and provides an interface to the upper level application for accessing the DLS through the DLS interface at the same time.

However, it could be also interesting to consider other application scenarios in which a node that wants to access the LS service is not aware of the underlying DHT and does not partecipate to the maintainance of the P2P DLS. As result, the above architecture is decoupled between nodes (DLS peers) that are aware of the DHT and nodes (DLS client) that are not.

The architecture of a generic DLS client is shown in Figure 4.

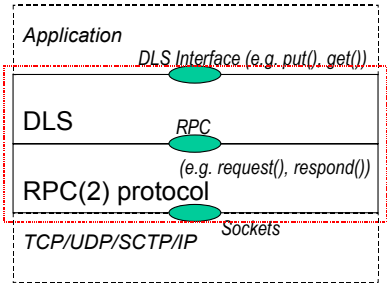

Figure 4 –DLS client (DHT-unaware) layered architecture.

In a DLS client the DLS layer still offers to the upper application layer a basic LS service (through the basic *put* and *get* methods). However, differently from a DLS peer, a the DLS layer within a DLS client maps directly these methods to proper RPC calls to a remote DLS server.

In general, such RPC protocol could be different from that used by the Peer layer within DLS peers, and for this reason is here referred as RPC(2) (Figure 4).

In order to effectivelly allow a DLS client to access the DLS, a sort of DLS adapter peer is required. Such adapter peer in adjuction to the normal peer operations should perform DLS server function, that is a sort of relay function allowing DLS client requests to be relaid to the P2P DLS system.

The overall architecture of a DLS client and a DLS peer adapter is shown in Figure 5.

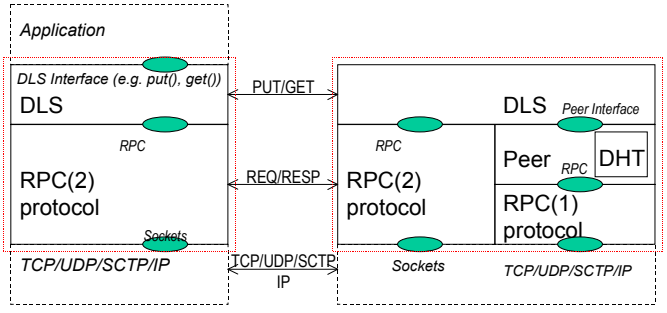

Figure 5 –DLS client with adapter peer.

Note that RPC(1) and RPC(2) may or may not be the same protocols; this is just an implementation issue.

It is important to remark that the proposed layered architecture is general and independent from the selected DHT algorithm and RPC protocols. More precisely, it is defined with the following three components:

- a DHT algorithm;
- a RPC(1) protocol used for managing the DHT (inserting a new peer, updating the DHT, etc.);
- a RPC(2) protocol used to perform basic LS queries like put(), get() on the distributed LS, used by non-DHT peers; since DHT peers uses for maintaining the DHT, protocol C3 is intended for pure DHT access at the border of the P2P system.

3. IMPLEMENTATION

We have implemented a Java-based complete framework to realize a DLS with any DHT algorithm and suitable communication (RPC) protocol. Our implementation therefore does not make any assumption about the DLS components used and follows the architecture sketched above.

Our implementation is in accord to the architecture described in the previous section and represented in Figure 2 and Figure 3. A client application that uses such DLS would use the service by calling two basic DLS-access API methods: *put()* and *get()*. The DLS layer would then store or retrieve the requested resources through the proper use of the underlying layers.

In order to be protocol-independent, DHT messages are handled internally by the peer through neutral message objects that contain the basic information about any request or response that might occur within the DHT. However, these objects are not strictly defined, that is, extra information that might be significant for a specific DHT algorithm can be added and retrieved in a transparent way.

Obviously, also the DHT algorithm logic is not specified. The peer's core the implements the DHT logic offers the basic functionalities that are common to all the DHT algorithms. The specific logic must be implemented by extending the abstract Peer class.

DHT API methods are asynchronous, that is, once they are called they are executed on a different thread, thus allowing the peer to continue its lifecycle and be able to handle incoming requests. Once the method has been executed, the listener that was registered for this event (usually the peer) is notified through callback methods. However it is possible, if needed, to implement synchronous methods from asynchronous ones simply by blocking the main thread until the callback method is called.

The DLS framework that we have realized has been successfully used to implement an actual Java-based DLS. The DLS supports the Chord and Kademlia DHT algorithms and uses dSIP [3] as a communication protocol within the overlay.

The implementation process has been focused on the extension of the abstract Peer class in order to realize the DHT algorithms, together with some other classes that implement the DHT RPCs. A DHT Communicator has also been implemented in order to support dSIP as a communication protocol. This step has simply required the creation of the classes to parse the incoming the dSIP messages and transform them into the neutral message representation (DSIPMessageParser) and to create new dSIP messages (DSIPMessageFactory). The DHT Communicator has been based on the MjSIP stack [10], which provides the SIP stack and the API for the SIP signalling protocol [5].

Our DLS is used as a general purpose DLS, which means that the resources stored into the DHT do not refer to a specific URI scheme. In this way the DLS can be used by many applications that work with different URI schemes. An important side effect of merging different location services into a unique DLS is to strengthen the LS, thus allowing for high availability and fault tolerance.

Figure 6 shows the DLS implementation structure, which is totally compliant to the architecture that was described in the previous sections.

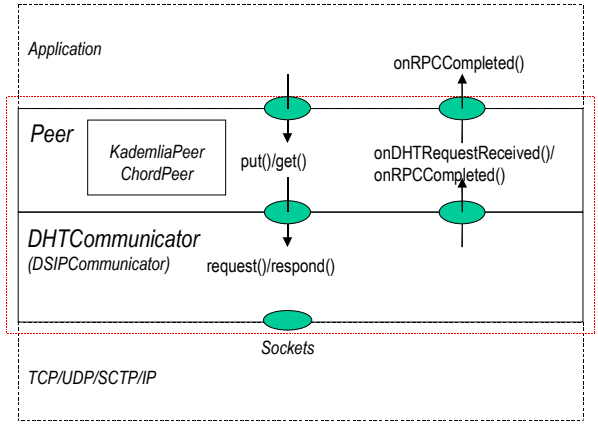

Figure 6 – DLS framework implementation.

We have also implemented, according to Figure 5, the protocol adapters for the SIP and HTTP protocols, so that a peer can receive and process SIP and HTTP requests sent by legacy applications that are not aware of the P2P substrate.

4. SAMPLE APPLICATIONS

Our DLS has been used to create some sample applications that show how the DLS can be exploited to create purely distributed applications.

4.1. Peer-to-peer SIP calls

Pure P2P SIP calls can be performed by exploiting the DLS as a SIP LS. Legacy SIP User Agents would register themselves using a peer enrolled into the DHT as a Proxy server. The peer would receive the registration request at its SIP protocol adapter interface and store the UA's contact into the DHT. When a UA wants to perform a SIP call, it sends an INVITE request to the peer for some user. The peer performs a lookup to resolve the target user's address and retrieve its

location, then it would forward the INVITE request to the UA. The SIP session is now initiated. User registration scenario is sketched in Figure 7. Session establishment is shown in Figure 8.

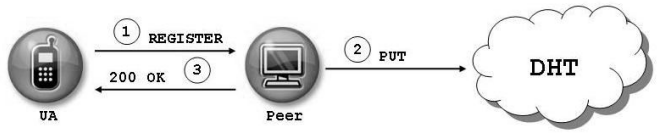

Figure 7 - Peer-to-peer SIP user registration.

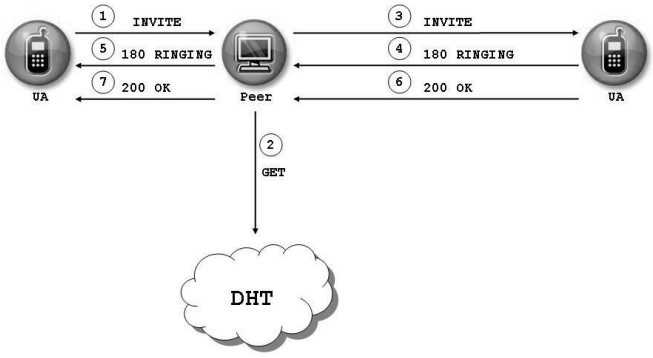

Figure 8 - Peer-to-peer SIP call.

4.2. Virtual HTTP web server

Another DLS-based application is a HTTP distributed virtual server. In such application, a virtual web server is deployed. This means that the files are not stored on the same host but are published by a number of nodes that collaborate. Data replication can be achieved by storing the same information on different hosts, and the website contents can be partitioned among several nodes.

4.3. Distributed database

RFC 2397 defines a method for mapping any short data within a standard URI (data:). Such URIs encapsulate small amount of data, such as small images. We could think to extend the usage of data: URIs to store some short data of any kind into the DHT. It is important to point out that the data should be short, because DHT's self-reorganization requires the transfer of resource information among the peers and, since this happens quite often in real P2P networks, it would cause an overload of the network traffic. However, under the hypothesis of short data, the DLS can be extended to become also a distributed database. Note that, in this case, the data are self-contained in the URI and the information stored in the DHT are not an access information but the data itself. Therefore, the data stored into the DLS are persistent, that is, they can still be accessed even when the node that has published them leaves.

5. CONCLUSIONS

In this paper we have presented the architecture for a Distributed Location Service. A DLS is composed of three main components: a DHT algorithm, a P2P protocol, and a client protocol. We also have presented a Java-based framework to build a DLS independent from the DHT algorithm and communication protocol to be used transparently by any application. Our implementation also provides an actual DLS that uses Chord and Kademlia as DHT algorithm and dSIP as a communication protocol. We also realized the protocol adapters for the SIP and HTTP protocols in order to let the peers receive and process messages sent by legacy applications that are not aware of the P2P substrate. Finally, we have presented some sample applications to show how the DLS can be exploited to build distributed applications.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. 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, February 2003.

[2] P. Maymounkov and D. Mazires. Kademlia: A Peer-to-Peer Information System Based on the XOR metric. In 1st International Workshop on Peer-to-peer Systems, 2002.

[3] D. Bryan. dSIP: A P2P Approach to SIP Registration and Resource Location. Internet-Draft draft-bryan-p2psip-dsip-00, IETF, February 2007.

[4] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax," January 2005, status: IETF Standard Track.

[5] J. Rosenberg, H. Schulzrinne, G. Camarillo, J. Peterson, A. Johnston, and E. Schooler, "RFC 3261: SIP: Session Initiation Protocol," June 2002, status: IETF Standard Track.

[6] L. Masinter, "RFC 2397: The "data" URL scheme," August 1998, status: IETF Standard Track.

[7] M. Zangrilli and D. Bryan. A Chord-based DHT for Resource Lookup in P2PSIP. Internet-Draft draft-zangrilli-p2psip-dsip-dhtchord-00, IETF, February 2007.

[8] S. Cirani and L. Veltri. A Kademlia-based DHT for Resource Lookup in P2PSIP. Internet-Draft draft-cirani-p2psip-dsip-dhtkademlia-00, IETF, October 2007.

[9] S. Cirani. Implementation of the Chord and Kademlia DHTs with dSIP, October 2007. http://www.mjsip.org/projects/p2psip/p2psip_dsip_071025.zip

[10] L. Veltri. mjSIP project, 2007. http://www.mjsip.org