# Range-capable Distributed Hash Tables

Alessandro Soro
CRS4
POLARIS
Edificio 1, 09010 PULA (CA - Italy)
+39 070 9250 261
asoro@crs4.it

Cristian Lai
CRS4
POLARIS
Edificio 1, 09010 PULA (CA - Italy)
clai@crs4.it

## ABSTRACT
In this paper, we present a novel indexing data structure called RDHT (Range capable Distributed Hash Table) derived from skip lists and specifically designed for storing and retrieving geographic data from a structured P2P network overlay. We have developed RDHTs as backend for the DART search engine, whose goal is to efficiently answer complex queries based on semantics and geographical context of the information stored in a P2P network. Queries are "range enabled", in the sense opposite of the exact matching. Range and semantic queries on location based resources make it possible to answer questions such as "Where is the nearest bookshop?". RDHTs merge the robustness and scalability of distributed hash tables with the simplicity and self maintenance of skip lists, while providing efficient support for range queries and proximity queries.

## Categories and Subject Descriptors
E.1 [**Data Structures**]: Distributed Data Structures

## General Terms
Algorithms, Performance.

## Keywords
Distributed Hash Table, Peer to Peer, Search Engine.

## 1. INTRODUCTION
Distributed Hash Tables are emerging as a key technology in P2P applications as a consequence of their robustness and scalability.

Several projects[4][7], as well as popular file sharing applications[1][8] make use of DHTs in order to distribute the data over a large number of *peers*, that contribute storage to a *community* of users. In the last years the research and the development in the P2P field has been considerable. Napster, Gnutella2, Edonkey2K, Bit Torrent, Kademlia [15] are only a few examples of consolidated protocols/architectures. The use of a such shaped infrastructure in general is justified due to the most relevant features of P2P systems, such as resistance to the censorship, decentralization, scalability, security, etc. To distribute data among thousand or million of peer not only means to have a huge amount of information, but also means to make confidence to a robust system free of restriction from a centrali authority.

Unfortunately DHTs only support exact matching queries and thus are not practical to support non trivial applications like geographic or location based services. Indeed, a typical query to a geographic based search engine will specify a geographical position and a range, for example 'find hotels within 10 miles from my current position'.

The work illustrated in this paper has been conducted as part of the DART[6] search engine. The project is focused on realize a distribute architecture for semantics searches on the Web and the access to personalized contents. Also the project aims at supply position based information strictly related to a user indicated area, so to provide spatial queries based on geo-referenced data.

An important aspect of the DART project is to stimulate the birth of a community of users that contribute storage and CPU cycles to the creation of a global, *public* index of Internet resources, strongly centered on personalized content. Such distributed index should support several applications, spacing over texts analysis, retrieval of semantic labeled documents, query of spatial referenced information, etc. While the storage system required for this applications *must* be distributed to put it behind any possibility of censorship or centralized control, the possibility of efficiently execute range queries must be granted.

In the following sections we present a novel indexing data structure called RDHT (Range capable Distributed Hash Table) derived from skip lists and specifically designed for storing and retrieving geographic data from a structured P2P overlay network, intended as distributed geographical information systems.

The DART project has been partially funded by the Italian Ministry of University and Scientific Research, contract grant number 11582

## 2. SCENARIO
The scenario we refer to when designing the DART Network Overlay consists of a location based service that spots items in a distributed DB, based on the geographical position of the user. Users act both as information providers and consumers: for example a GPS-enabled computer installed on a car or on a boat will query the network for possible dangers within a given range from its position. At the same time it stores on the network information about potential dangers that it can register along the path, based on simple events that can easily be translated in rules, such as: the activation of the airbag implies a potential danger for


*Conference'04*, Month 1–2, 2004, City, State, Country.

other vehicles that are approaching my geographical position. This scenario presumes the ability of clients to run queries based on the geographical position, interval and a class of entities, but, as a starting point, we have reduced the problem to that of querying a distributed DB that stores sorted lists of integer values.

The first prototype we have implemented shows a web interface based on Google Maps API [9], which allows the user to select a

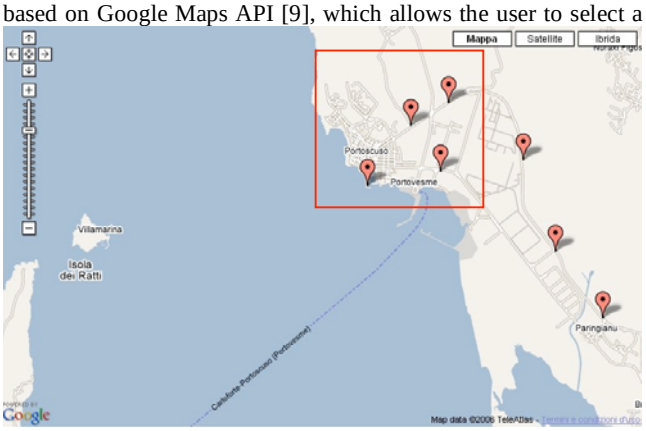


**Figure 1**

region and to submit query to the system on the specified range.

In Figure 1 is shown the response to the query bounds by the red rectangle.

## 3. DHT for Geographical IR

The DART research project is aimed at designing an architecture and deploying a toolkit to store and retrieve a global, public index of Internet location based resources. For simplicity at the moment we consider only GPS coordinates. DART users are supposed to contribute to the system in terms of storage and CPU cycles, but also sharing information as P2P applications use to share assets. As a backend to DART, an efficient, robust and scalable distributed filesystem is required and Distributed Hash Tables over a P2P overlay have extensively proven to meet these requirements.

The most important features related to DHTs can in fact be summarized as follow:

- *efficiency and scalability*, the number of messages exchanged to route a query to its destination is O(log(N)), where N is the total number of nodes;
- *no maintenance*, no administrative operations are required, no central authority or complex process is required to maintain, balance or fix the distributed data structure;
- *simplicity*, the algorithms behind distributed hash tables are relatively simple to understand and implement;
- *robustness*, the ability to survive massive failures is a key aspect when deploying largely distributed applications, such as file sharing applications;

DHTs can store and retrieve efficiently a huge amount of information, but queries require an exact knowledge of the resource ID (the key).

This excludes many applications, in which the key is known only approximately, e.g. a geographical position, or is known to fall within a range e.g. a time interval. The DART project defines a ***DART Network Overlay***, whose goal is to support a wide variety of distributed applications, by providing a flexible, efficient, and robust distributed filesystem, capable of range queries.

We call this filesystem RDHT (Range capable Distributed Hash Table): its basic idea is derived from skip lists[16], though in RDHT we lose the concept of different *levels* of pointers, in

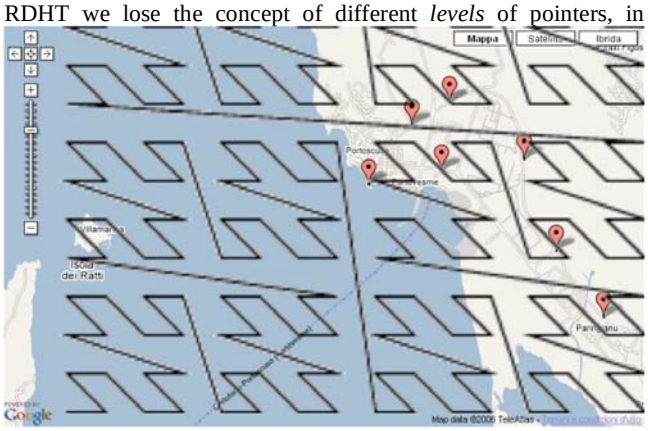


**Figure 2**

exchange for a self organized *backbone*. The storage of index information, as well as any other operation, is built on top of the Kademlia protocol.

Note that range queries only make sense if the items stored can be ordered with respect to one or more attributes. RDHTs store simple integer values. The transformation from Objects attributes to integer values is application specific: for certain applications a lexicographic order, resulting in a list can be applied. In geographic applications linearization is often used to represent n-dimensional coordinates. **Error! Reference source not found.** shows the way geographical coordinates can be linearized using a z-curve [11], i.e. interleaving the bits of the X and Y (or longitude and latitude) coordinates of the spots to obtain a single integer value; Items' coordinates are represented as an integer value, that is a point in the z-curve, and the limits of a query for a rectangular region are translated to a linear interval of the z-curve itself. This greatly simplifies the store and retrieve operations, reducing the problem to that of storing and querying sorted integer values. Note that in contrast to lexicographic ordering, points that are close in the map get generally translated to close integer values, anomalies must be filtered from the result set.

## 4. RANGE-CAPABLE DHTs

The basic idea behind RDHTs is to use the underlying DHT to store index information as well as the data themselves. This is analogous to other approaches, for example [17], but, RDHTs require less effort in organizing and maintaining the index.

Whenever a new value V is inserted in the RDHT, two new pointers are registered, bounding V to its nearest neighbors succ(V) and pred(V). In this way a chain of pointers get stored in the DHT to guide lookup operations. Index information for value V is stored under the key hash(V). For example: from the initial list represented in Figure 331.

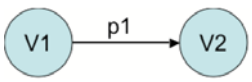


**Figure 3**

where the arrow *p1* represents index information stored under the key hash(*V1*). Inserting the new item *V3*, whose value is greater than *V1* and less than *V2* will result in the configuration shown in Figure 442:

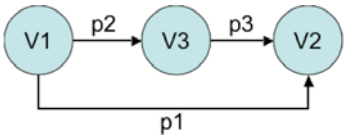


**Figure 4**

where the pointer *p1* remains valid, but in addition the pointers *p2* and *p3* get stored. In this way older values and pointers are used as a backbone to guide lookup operations: as new values are stored, pointer *p1* get *stretched* and can be exploited to make long *jumps* skipping large portions of the RDHTs. The result of inserting the new values V4 (V3 < V4 < V2) and then V5 (V3 < V5 < V4) is shown in Figure 553 and Figure 664.

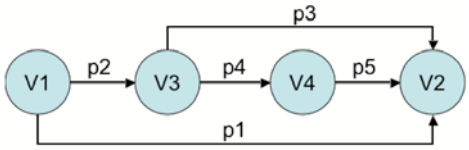


**Figure 5**

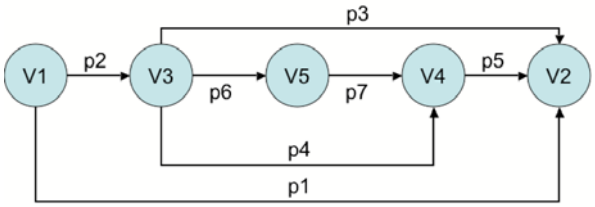


**Figure 6**

Again the pointer *p3* and *p4* have been stretched and four new pointers have been inserted: *p4, p5, p6* and *p7*. Note that no removal or update is necessary when inserting a new value. Old pointers remain valid in that they don't link each item to its *next* but only represent a predecessor/successor relationship. Each insert operation requires storing a fixed amount of information, so the total storage required by the index structure for a list of N items is O(N).

## 4.1 Primitive Operations

Primitive operations require exchanging a moderate amount of messages. A so shaped data structure requires basic primitive operations, such as lookup, nearest, insert and range. To efficiently implement nearest and lookup operations, backward pointers must be stored in addition to forward pointers illustrated so far. Low level primitives relative to the DHT operations, like *join*, *ping*, ecc. are implemented in the Kademlia network overlay, and will not be described here.

***Lookup*** means discover an item stored in the data structure. It starts from a known value and executes several lookup operations on the underlying DHT, in order to fetch index information. For example to lookup the item V4, starting at item V1, the systems fetches all the pointers stored under key hash(V1), chooses the longest possible jump that don't overshoots V4 and iterates, until a pointer to V4 is found. If no such pointer exists the lookup operation will fail. Of course using a fixed starting points for lookup operations constitutes a bottleneck and a single point of failure, unacceptable for our application. This can be avoided by storing backward pointers so that a lookup operation can start at any known item (chosen at random) and proceed forward or backward as needed.

The items (Vn- and Vn+) ***nearest*** to a given value Vn can be found executing a lookup operation to Vn. If Vn doesn't exist the lookup fails after retrieving Vn- and Vn+, the shortest pointer registered under hash(Vn-). If Vn exists, Vn- is the shortest backward jump and Vn+ the shortest forward jump registerd under hash(Vn).

To ***insert*** a new value Vn the system must at first execute a ***nearest*** operation. The operation will fetch the items Vn- and Vn+ that must be linked to insert the new one. When an insert is performed the system stores four pointers that link Vn bidirectionally to Vn- and Vn+. Additionally a long pointer can be stored that links the item Vn to the item where the lookup started from. With this addition long pointers will be distributed equally over the population of the RDHT, and older pointers will not be overloaded with queries.

With these primitives a ***range*** query in the interval (lb, ub) can be executed quite trivially through a succession of *lookup* and *nearest* operations, that step through the RDHT forward or backward, beginning at the lower or upper bound of the interval. Alternatively a range query can be executed searching for the items *nearest* to the *median(lb, ub)*, and then repeating this operation recursively over the two subintervals until no new element is discovered or a given grain is reached. Note that this second strategy can be easily parallelized.

Note that there is no ***remove*** primitive, once stored an item cannot be deleted. This implies that malicious removal of data is not possible. How this affects the performances of the system, the flexibility of the protocol and the semantics of the primitives is beyond the scope of this paper.

# 5. IMPLEMENTATION

We have developed a prototype implementation of the DART Network overlay, written in Python and based on Khashmir[12], an implementation of the Kademlia protocol.

Preliminary measurements, executed in a test environment, show the goodness of the architecture. As illustrated in Figure 5, with a RDHT of 10000 items, distributed across 100 peers, we need between 6 and 8 iterations to complete a store/lookup operation, each iteration consisting of a Kademlia *lookup* that in turn requires ~log(100) messages to succeed. At the moment we


**Figure 7**

obtain this results with items generated and stored randomly.

## 5.1 Known Issues

The efficiency of Insert and Lookup operations strongly depends on the values to be inserted (almost) in random order. Our insight is that the case of a long sequence of consecutive values inserted in normal operation is very unlikely to happen for two reasons: first of all the system is intended to be used concurrently by many peers, that continuously store values in the RDHT; additionally the linearization algorithm will scramble the values stored (referring to **Error! Reference source not found.**, an item moving in the map will not store sequential values, unless it follows exactly the z-curve).The system is still a prototype, deep testing in a truly distributed environment and under full load is still to be done. Particular attention will be paid to load balancing, both with respect to storage distribution and query resolution effort.

# 6. RELATED WORK

The application of the P2P paradigm to GIS is targeted at getting rid of the typical problems of centralized systems, such as network overload, single point of failure, censorship, lack of scalability. Community oriented architectures for geographic based services, often referred to as *geocollaboration* frameworks, such as DART are an open and emerging field in distributed computing and GIS, see for example[2][3][10][13][14], and their applications spread from emergency management to amusement.

Several solutions have been proposed to face the problem of range queries over P2P systems, each one with its own strengths and weaknesses. Unstructured P2P systems address the problem of range flooding queries to all peers in the network, thus requiring O(N) messages. More scalable solutions require to store a distributed indexing data structure in the P2P network itself, and use this to guide range queries. P-Trees [5] consist of distributing a B+ tree on a P2P network, storing at each peer a fraction of the overall tree. The integrity of the P-Tree is granted by periodically executing a *ping/stabilization* process, that checks and recovers nodes from failure.

Prefix Hash Trees [17] represent an efficient a robust distributed data structure and can be implemented over a generic DHT without modifying the routing algorithms, but the *insert* and *delete* operations can result in splitting/merging nodes, thus requiring complex balancing operations. PHTs have been applied to location based services in Placelab[13], a framework for device positioning using a distributed index of radio beacons.

# 7. Ongoing activities

The challenge to make range queries in distributed systems is becoming a need due the direction followed by information retrieval research. The preliminary bases crated by this work are growing toward information based architectures strictly centered on location based features.

Most of the obstacles encountered up to now impose to consider the kind of information indexed in a such structured architecture. The case of integer values (or integer assimilated values in the case of GPS coordinates) is of course a simplification. Most interesting is take into account more structured information tagged with metadata. Eventually will be interesting to analyze results on research in range queries in semantic arranged information.

# 8. REFERENCES


[1] Azureus - Java BitTorrent Client. http://azureus.sourceforge.net/

[2] Balram, S., Dragicevic, S. Collaborative Geographic Information Systems: Origins, Boundaries and Structures.

[3] Carboni, D., Sanna, S., Zanarini, P., GeoPix: Image Retrieval on the Geo Web, from Camera Click to Mouse Click. *In Proceedings of MobileHCI'06*, September 12–15, 2006, Helsinki, Finland, ACM Press.

[4] Chawathe, Y., LaMarca, A., Ramabhadhran, S., Ratnasamy, S., Hellerstein, J., Shenker, S., A Case Study in Building Layered DHT Applications. *IRS-TR-05-001* Jan 2005

[5] Crainiceanu, A., et Al Querying Peer-toPeer Networks Using P-Trees. *In Proceedings of WebDB Workshop* (2004).

[6] DART – Distributed Agent Based Retrieval Tools. Soro A., Paddeu G. and Armano G (eds.) (to be published)

[7] Druschel, P., and Rowstron, A. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility. *In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)* Lake Louise, AB, Canada, October 2001.

[8] eDonkey2000 - Overnet. http://www.edonkey2000.com/

[9] Google Maps, http://maps.google.com/

[10] Guan, J.H., Zhou, S.G., Wang, L.C., Bian, F.L., Peer to Peer Based GIS Web Services. *Proceedings of the XXth ISPRS Congress*, July 2004 Istanbul, Turkey

[11] Jagadish, H. V. Linear clustering of objects with multiple attributes. *In Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'90)* May 1990, pp. 332–342.

[12] Khashmir. http://khashmir.sourceforge.net/

[13] Lamarca et. Al. Place lab: Device positioning using radio beacons in the wild. *Technical Report IRS-TR-04-016, Intel Research Seattle*. Sept. 2004.

[14] MacEachren, A. M., Cai, G., Sharma, R., Rauschert, I., Brewer, I., Bolelli, L., Shaparenko, B., Fuhrmann, S., Wang, H. Enabling Collaborative Geoinformation Access and Decision-Making Through a Natural, Multimodal Interface. *International Journal of Geographical Information Science*

[15] Mayamounkov, P. Maziers, D. Kademlia: A peer-to-peer information system based on the xor metric. *In Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. 2002.

[16] Pugh, W., Skip Lists: A probabilistic alternative to Balanced Trees. *Workshop on Algorithms and Data Structures*. 1990

[17] Ramabhadran, S., Ratnasamy, S., Hellerstein, J., Shenker, S Prefix Hash Tree - An Indexing Data Structure over Distributed Hash Tables. 2004