GTAG: Supporting Geographical Queries onto DHTs

Jordi Pujol Ahulló, Marc Sànchez Artigas, Pedro García López

Universitat Rovira i Virgili

Computer Science Engineering and Mathematics Department

Av. Paisos Catalans, 26

43007 Tarragona

Tarragona, Spain

{jordi.pujol,marc.sanchez,pedro.garcia}@urv.cat

Antonio F. Gómez Skarmeta

Universidad de Murcia

Information and Communication Engineering Department

Campus Universitario de Espinardo

30100 Murcia

Murcia, Spain

skarmeta@dif.um.es

Abstract

Location-based services (LBS) are currently receiving world-wide attention as a consequence of the massive usage of mobile devices, but such location services require scalable distributed infrastructures in order to resolve spatial queries efficiently. We propose a novel methodology to enable geographical query support to distributed hash tables (DHTs). The contributions of our methodology are the followings: a) our technique is DHT-generic, b) it makes an effective clusterization of nodes and information into geographical areas, c) providing data locality without sacrificing routing and data load balancing, d) it is able to answer classical spatial range queries, as well as e) a new kind of queries we call geocast, all of them in a distributed, scalable way. We demonstrate the feasibility of our approach by means of meaningful simulations.

1 Introduction

Later peer-to-peer systems are classified into two categories: unstructured and structured peer-to-peer systems. The former category includes systems like Gnutella [1], where lookups are performed by a flooding approach and, therefore, these systems can report false misses[1]. This fact becomes prohibitive for large-scale systems, so structured peer-to-peer systems appeared (also known as distributed hash tables (DHTs) [15, 22, 26, 34]). These systems perform lookups in an efficient *logaritmic cost* in terms of node hops according to the current number of nodes in the network. Moreover, DHTs do not report false misses and, thus, they guarantee that if a certain object *ob* exists in the system, it will be found. It is a necessary but no sufficient property in modern (distributed) applications.

The importance of geographical information systems and location-based services has considerably increased in the last years. This boom is a direct consequence of the proliferation of mobile devices, ubiquitous networking

---

[1]Answering object *ob* as not found in a system that actually has *ob* is a false miss.

and positioning systems. Examples of such location-aware services include querying for specific resources in a geographic area or even integrating information collected by sensors in a given region. In this line, Google Local or Yahoo Local are centralized repositories that store location information and permit spatial queries using visual map interfaces. But we are interested in achieving a scalable, distributed solution.

Geographical information, business organization or knowledge fields are clear examples of (semantically) structured information. All they have in common their organizational structure. For instance, we can define that world geographical information is categorized within continent (disjoint) geographical areas; continent information is itself organized into country (disjoint) geographical areas; and so on until the necessary level of geographical precision is reached. It is clear also that the union of all geographical areas from a certain level form a higher level geographical area. For instance, all countries of a continent form the continent itself. Additionally, we can define a naive rule that information is stored into the system just into the smallest geographical covering area (i.e. into the lowest geographical covering precision level). Thus, every data managed by the system is always related to a unique area. For example, information related together to different M countries of the same C continent is sufficient to be stored at their C continent level, instead of storing M copies of the same data within every country. Therefore, we improve the system performance by avoiding unnecessary overhead.

It is obvious that this kind of information has a *hierarchical structure* behind it, *clustering* related information into the same area (level). We believe that a hierarchical substrate of clusters will greatly deal with the related complexity on any hierarhical information organization in distributed systems. As a proof of concept, we apply our solution to the geograhical information field, but one can easily apply our approach to any other information field with the above properties, by replacing in the rest of the paper the geographical terms by the target information field accordingly. Thus, because we are interested in giving a generic solution to DHTs, instead of solving it for specific DHTs, *we introduce in this paper a DHT-generic methodology to get geographical information efficiently clustered and, in this way, perform efficiently spatial range queries over the system*. We have also designed a new kind of aggregate lookup so-called *geocast*, that *retrieves efficiently related information from the different local geographic levels* leveraging the clustered peer-to-peer substrate.

The aforementioned DHTs are basicly characterized by a uniformly distributed at random set of nodes througout usually a unidimensional keyspace, by the use of some uniform hashing function like SHA-1. While the same technique is applied to distribute data throughout the same keyspace to achieve *data load balancing*, this fact enters clearly in conflict with data organization to perform efficiently range queries, because these uniform hashing functions destroy the semantic proximity (i.e. *data locality*) [7]. Therefore, some kind of locality- or order-preserving hashing function (LPHF or OPHF respectively) is required in order to retain this proximity once data is managed by the system (e.g. Space Filling Curves (SFCs) [23]). It is easy to see the *trade-off* presented to the designer: how much data locality is preserved versus how much data load balancing is retained. If the adopted solution retains the data locality (what it would be desirable), then there will not be data load balancing between nodes and, therefore, some external technique must be applied to balance the data load. Instead, if the solution ensures data load balancing (as in DHTs), then this solution will not preserve the semantic proximity and

performing higher level operations (like range queries) will be very expensive [20].

From another viewpoint, we assume a geographical information system as a *two-dimensional* problem solving. *Geographical location*, defining the geographical area where the information is related, and *keywords*, describing the kind of resource available, are the two variables that this kind of systems must address. In the rest of the paper, we genericly call these two variables *geotags* and *tags*, respectively. $< geotag : Spain, tag : capital >: Madrid$ is a clear example of geographical data, formed by a pair of the form $< key >: value$. Thus, the system must perform in an efficient way insertions and searches of information by declaring both geotag and tag. To give an example, one can imagine a Google Local user looking for (tag:) *Italian restaurant*s in (geotag:) *Madrid*. And the user wants a delay-less answer with *related* results. In fact, these related results must be measured by the system with some proximity function, evaluating geographic and semantic proximity regarding to the given geotag and tag, respectively, in order to retrieve the most *interesting* information to the user.

Overall, performing efficiently such geographical range queries over DHTs that naturally do not support this kind of non-trivial higher level operations is a complex problem and we foresee the following points (which appear in no specific order) as the main challenges to deal with:

- Manage multi-dimensional data domains over DHTs that work with unidimensional keyspaces.

- Develop a DHT-generic approach to deal with hierarchically clustered data domains.

- Define some kind of mapping scheme (e.g. OPHF) to retain semantic proximity when data is stored, maintaining data load balancing.

- Define the related algorithms to perform efficiently range queries over hierarchically clustered multi-dimensional data domains.

To the best of our knowledge, this work is the first in such a DHT-generic clustered proposal for an efficient organization of geographical information into a distributed peer-to-peer system and, thus, our main contributions are the following ones:

- We map the location information into the *suffix* of the node identifier (node ID), leaving the node ID preffix available for application-specific uses and providing at the same time *efficient routing* through the different geographical clusters (*geoclusters*). We call this suffix part the *clusterId*. As our approach is based on the information mapping into the node ID, regardless the specific DHT routing tables, we promote a DHT-generic approach.

- On the contrary to what could be initially expected, our approach provides *data locality* without sacrificing the overall node load balancing and DHT routing properties for skewed node IDs. Furthermore, our system provides *data load balancing*, enhanced by the use of efficient cluster-based caching schemes that cannot be achieved in flat-based overlays.

3

- Our system supports efficiently *geographical range queries* combining both geotags and tags. Geotags are encoded in the *clusterId* and tags are stored in the appropriate cluster.

- Finally, we also present a novel search abstraction named *geocast queries*. Inspired in the anycast proximity primitive, we enable *efficient local-to-global queries* (e.g. geocast tag:java) that returns information in increasing order of geographical area (specificly region, country, continent and world), leveraging the clustered peer-to-peer substrate and obtaining better performance results than in other flat aproaches.

The rest of the paper is organized as follows: Section 2 makes a review in the related work; Section 3 states the problem we deal with in the current paper; we then apply our DHT-generic technique over Symphony [15] in Section 4 and describe how location information is embedded into the *clusterId*; in Section 5 we describe how our approach balances both routing and data loads; in Section 6 we analyse the higher level query types that our approach supports and Section 7 concludes the paper.

2 Survey

In this paper, we are mainly interested in decentralized DHT-generic infrastructures offering scalable, large-scale higher level location services like spatial range and approximate queries. In fact, a lot of work have been devoted to architectures for supporting such geographical information systems. Most of these architectures rely on overlays of *stable nodes* connected to the wired network infrastructure. In this case, authors assume non-volatile environments with small churn that are quite appropriate for structured peer-to-peer overlays.

Nevertheless, supporting geographical range queries over structured peer-to-peer overlays is still an open research problem that implies strong challenges and a trade-off between them: *geographic information organization*, *multi-dimensional indexing*, *data locality*, and *data and routing load balancing*.

Geographic information organization. Given the characterization seen in the introduction, geographic information is naturally hierarchical, grouping information by disjoint areas at different levels of precision (i.e. a certain set of geographic regions form a country, a certain set of countries form a continent, and the whole set of continents are the world), where range queries should be efficiently performed on every level. We believe that existing work does not cope the essence of the problem and (possibly) adds some kind of overhead.

A first approach is to build a logical layered tree on top of a flat DHT. For example, in Place Lab [6] authors apply linearization techniques (Z-curve [19]) to store information in a logical tree (PHT [20]). As explained in [33], a DHT lookup is needed for every tree edge starting from the root, thus resulting in very high costs of $O(\log^2(n))$ or even $O(\log\log(n) \cdot \log(n))$ for structures with balanced depth. Obviously, such logical trees are inefficient due to a lack of data locality.

Thus, why we do not use any tree-based existing DHT? Kademlia [16] is a structured peer-to-peer network that builds a tree, and routing is based on the XOR metric. This system routes to zones at certain distance from the initiator node, property used in [18] to locate neighbouring attachements points in Beyond 3G mobile networks. Unfortunately, Kademlia does not have the concept of *successor's node*, required to guarantee no false misses.
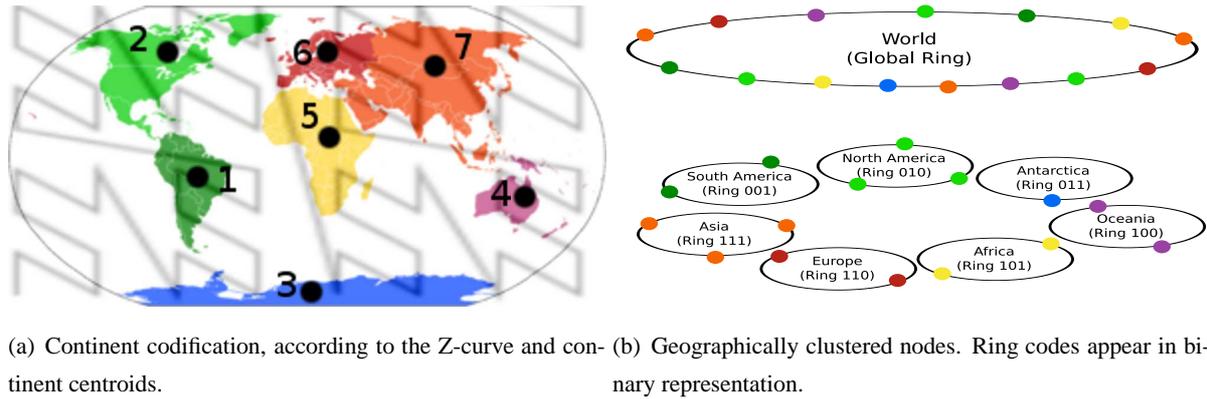
(a) Continent codification, according to the Z-curve and continent centroids.

(b) Geographically clustered nodes. Ring codes appear in binary representation.

Figure 1. Example of homogeneous hierarchical design, organizing nodes geographically.

P-Grid [3], a trie-based overlay, works in a similar way in [7]. In that work authors emulate a Skip Graph [5] behaviour in order to perform range queries. Unfortunately, both works have in common that route in a prefix-based way (differing from our suffix-based approach), and that builds just a tree substrate without the notion of node and information clustering, necessary condition to achieve data locality.

Another interesting trend in spatial databases is to adapt well-known structures like the R-tree to a peer-to-peer cluster architecture [17, 28]. If the design of an overlay inherently supports a hierarchy of clusters, the mapping for spatial data is straightforward. Nevertheless, the existing approaches in this direction are still immature and do not address high scalability, churn and consistent routing and load balancing.

We can differ hierarchical peer-to-peer systems by their design: *heterogeneous* or *homogeneous*. A *heterogenous* hierarchical design [25, 30] usually distributes all nodes in leaf clusters (i.e. in the lowest level of the hierarchy) and selects some peers (not all of them) to participate in the inmediate higher level of the hierarchy, and so on until the hierarchy is completed by some criteria (e.g. a maximum threshold in the number of nodes per cluster [11]). Instead, a *homogeneous* hierarchical design makes to participate every node in all levels it pertains (see Fig. 1). Thus, it is easy to see that on every level of the hierarchy we can find all nodes in one or more disjoint clusters [24]. Intuitively, while it is sufficient on homogeneous designs to locate the related cluster and then apply some algorithm to perform efficiently geographical range queries specifically only on that cluster, the heterogeneous design makes this operation more complex. In fact, super-peers that receive such kind of query should contact to some leaf clusters in order to apply some algorithm to perform the range query, adding extra hops and w.h.p. lack of geographical node organization, becoming even more complex the range query algorithm (e.g. [14]). Thus, we believe that a homogeneous design of a hierarchically clustered peer-to-peer system is able to deal successfully with this challenge in a distributed fashion. The idea is to *imitate the same geographic information organizational structure*, in order to achieve a desirable high level of scalability and performance.

Multi-dimensional indexing. Geographical queries are in fact a special case of multidimensional data indexing. Whereas R-tree, KD-trees and quadtrees are well established multidimensional main-memory data structures

in centralized spatial databases, it is not clear how to efficiently support them in distributed repositories. A first approach [36] is to fully replicate the index and thus create a global index structure for all the repositories. This approach works for medium size networks but it does not properly scale for big networks. Other approaches are based on DHTs with multi-dimensional keyspaces (e.g. CAN [21], eCAN [29], SCAN [27]). Unfortunately, this family of CAN-based systems suffers of data and routing load unbalancing as we explain below.

Another trend is to reduce multidimensionality using LPHFs like Space Filling Curves (SFCs). SFCs have been intensively used in distributed geographical infrastructures. Such linearization techniques allow to store location information (i.e. coordinates) in unidimensional structures like DHTs. For example, Place Lab [6] system supports geographical range queries applying linearization techniques (in this case a Z-curve [19]) to store information in a logical tree (Prefix Hash Tree (PHT) [20]) built as a layered application on top of the DHT. The problem with logical layered tree structures like PHT is the absence of data locality in the system. This clearly worsen the overall efficiency of range queries, implying increased traffic and poor performance. Instead, our particular approach is based on the use of OPHFs, in order to reuse existing conventional DHTs that we do believe they have enough potential to support this kind of non-trivial higher level functionality.

Data locality. This property should guarantee that data is stored into the distributed system maintaining the semantic proximity, i.e. semantically close data items should appear in close nodes. To overcome the lack of data locality in DHTs, many authors have proposed an interesting technique for structured overlays that uses specific location-based node IDs. In [35] Shuheng Zhow *et al.* embed location information into the *node ID prefix*, reflecting the geographical hierarchy using *prefix clusterization*. In this way, authors achieve a routing improvement and obtain the desired data locality. This is however a risky approach: random node id assignment ensures uniform distribution of nodes in the overlay. If we renounce to consistent hashing, the overall load balancing of the system is seriously compromised. As a consequence, skewed node distributions can imply incorrect routing tables and hotspots in the network. Although authors in [35] aim to fix this problem, they need to use non-trivial ad-hoc techniques very sensible to dynamic changes in the system.

TOPLUS [10] also defines a hierarchy by means of node ID prefixes as in [35]. Nevertheless it is limited to the use of the 32-bit suffix of the identifier, representing the IP address. From this suffix, the clusterization is performed by fixing prefixes and inheriting the same disadvantages as above: routing and data load unbalancing. Instead, our approach embeds the location information into the *node ID suffix*, but achieving a high level on load balancing.

On the other hand, overlays based on data-centric Skip Graphs/Nets [5] provide data locality while ensuring at the same time routing load balancing. SkipNet [12] achieves that using two IDs: a randomly selected numericID and a contiguous nameID. It is thus possible to use location-based node IDs in Skipnet's nameID. In fact, we will compare our approach against SkipNet employing this *location-based nameID*, in order to take advantage of the SkipNet routing. For example, Brushwood [33] and SkipIndex [32] provide multidimensional indexing over tree structures constructed on top of a skip-graph-like overlay. The major drawback of such approaches is data load balancing: neither virtual node nor constrained load balancing techniques do really solve the aforementioned

problem for skewed data sets.

Routing and data load balancing. Routing load balancing enables the distributed system to route efficiently without hotspots. As conventional DHTs define the number of outgoing connections to other nodes, we measure the routing unbalancing in terms of the number of node incomming connections. To achieve routing load balancing, nodes should have the same number of incomming connections in average. Let us show a counterexample. Suppose a ring topology (like in Chord [26]) where most of the nodes are concentrated in a relative little ring sector, and few nodes are distributed throughout the rest big ring sector. In most of conventional DHTs, and in particular in Chord, it is completely an undesireble situation: Structurally such few nodes will be true hotspots, with a huge number of incomming connections and traffic. Instead, the concentrated nodes will receive very few incoming connections. For example, [35] suffers from routing load unbalancing.

In the same line, CAN [21], eCAN [29], SCAN [27] and GeoPeer [4] achieve node and data locality taking advantage of node IDs. Like before, in a geographically uneven node distribution, these systems balance neither data nor routing load. In fact, nodes in low dense regions will have a heavy incoming connectivity and traffic from high densely populated regions. Our aproach, instead, guarantees routing load balancing by enabling the default node ID construction, usually using the SHA-1 hashing function, and thus working in the desireble default DHT node configuration.

Data load balancing property should guarantee the same data load between nodes in average. Unfortunately, the above data-centric Skip Graph/Nets and tree-based approaches guarantee in some way data load balancing, but leave some of the challenges presented in this section open: *geographic information organization*, *multi-dimensional indexing*, *data locality*, and *data and routing load balancing*. Therefore, we will enfasize on the routing and data load balancing analysis in Section 5, instead of evaluating the cost of single searches. Remember that our approach is DHT-generic and, thus, leverages the selected peer-to-peer substrate properties.

3   Problem statement

In this paper, what we seek is a DHT-generic methodology to enable scalable, efficient geographical range queries so that:

1. Storage and retrieval operations (i.e. put/get, spatial range queries, etc.) can be performed at *distinct geographical granularities*,

2. While *conserving load balancing and routing properties* the original DHT supplies, regardless of the potential skewness on geographical information. In fact, information is not spread symmetrically around the world.

Additionally, to simplify the problem w.l.o.g, we define region, country and continent as the addressable geographical granularities, and we call *geocluster* to the set of nodes in such a granularity. Thus, this kind of *geographical query services* let users to search information at distinct geographical granularities. Even though there are systems that have solved this problem, they are centralized (e.g. Google Local).

Above all, to the best of our knowledge, our proposal is the first to build *a DHT-generic methodology that enables to look for interesting geographical information in large scale distributed systems, by embedding location information into the node ID, sacrificing neither data nor routing load balancing.*

In the following sections we detail our solution, which is equivalent to answer to the following questions:

1. How we can embed geographic information into the node ID, achieving a natural node affinity specialization?

2. How we can solve the above problem sacrificing neither data nor routing load balancing?

3. How higher-level queries (e.g. spatial range queries) can be nicely supported by such a resulting system?

4 The methodology

We propose to employ a *suffix-based assignment scheme to embed location information into the node ID*. The novel contribution of our approach is that our design leverages the peer-to-peer substrate structural properties and, while embedding such location information into the *clusterId* (i.e. the suffix node ID part), our design supports geographical queries at distinct granularities, grouping nodes on successively more specific geographical areas, and sacrificing neither routing nor data load balancing.

Specifically, we employ Cyclone [24], an algorithm for constructing homogeneous hierarchical DHTs, that nicely fits our suffix-based assignment scheme. As a case study, we detail how we apply the Cyclone algorithm onto Symphony [15] to produce what we call Geophony. We have selected in this case Symphony for its flexible selection of long links, that also enables an efficient routing in large scale networks with nodes non-uniformly distributed within the keyspace. Nevertheless, our approach is DHT-generic and, thus, *other DHTs* like Chord *can be used* instead of Symphony. Later on, we will describe how location information is embedded into the node ID.

4.1 Geophony: A geographically clustered DHT

We apply Cyclone [24] algorithm onto Symphony [15], producing a system we call Geophony, to reflect the hierarchical organization that naturally emerges in any geographical information system.

Cyclone is a generic technique to construct a homogeneous hierarchical DHT from a flat one, so that one can obtain the best of both worlds, without inheriting the disadvantages of either. This means that with the same number of links per node as in the flat DHT, the routing between any two nodes can be performed as efficiently as in the flat version. Its major virtue, however, lies in the utilization of node IDs to represent real-world hierarchies. To explain, each node is assigned a node ID from the range $[0, 2^b)$ so that the *cl* ($cl < b$) rightmost bits of its node ID (expressed in binary form) are used to encode its *clusterId* at level l (see Fig. 2). Recursively applying this strategy in all levels, one can easily produce a hierarchy, which can be defined as follows:

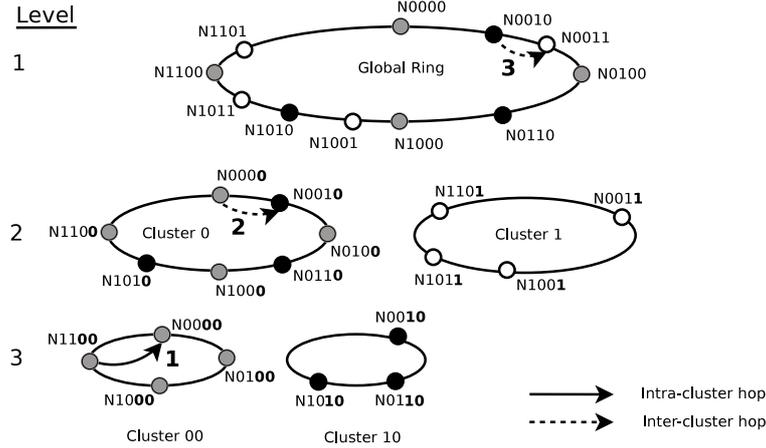1. In each level, all nodes are organized into disjoint clusters,

Figure 2. Hierarchy and (bottom-up) conventional routing **example when node** N1100 **sends a message with key** K0011**. On every cluster, absolute greedy routing is performed, discarding the** clusterId **bits accordingly at each level.**

2. There exits a universal cluster at level 1 that includes all peers (cl = 0), and

3. Each peer belongs simultaneously to at most η telescoping clusters (i.e., clusters of clusters of peers), with one in each level, where η denotes the hierarchy depth.

In particular, Geophony is similar to Symphony. Each node n creates $O(\log N) + K$ links (for some little constant K and the number of nodes in the system N) to other nodes. The two operands $O(\log N)$ and K in the cost evaluation are necessary to reflect the two kinds of routing a geographical information system has to provide. We call the former as (bottom-up) *conventional routing* and the latter as (top-down) *XOR routing* in clear reference to the way the routing is performed respectivelly: The former $O(\log N)$ conventional links enables conventional DHT routing, forwarding queries to those nodes responsible of performing (or starting) such operations; The latter K XOR links are selected by the XOR metric and are necessary to locate a certain geocluster (i.e. geographical area) where to perform some arbitrary user's operation (e.g. it is the equivalent to zoom in/out a map like in Google Local). Notice that some of the $O(\log N)$ and K links can be the same, thus reducing the total number of *different* links (see the case of node *N1010* in Fig. 3). In fact, given that K is constant, we can consider that each node maintains *different* $O(\log N)$ links. Additionally, we define every hop in the routing process as *intra-cluster hop* if both nodes pertain to the same (leaf) cluster or *inter-cluster hop* otherwise (see Fig. 2).

The $O(\log N)$ conventional links are chosen independently at random, so that the probability of node n choosing a node m as neighbour is inversely proportional to the distance from n to m. More specifically, each node n in Geophony maintains the following set of links for each cluster C_l in which n resides: a) s/2 successor and s/2 predecessor ring links and, b) $\lfloor \log |C_l| \rfloor$ long distance links for efficient routing within C_l, but retaining all the

9

links from the lower level clusters, where $|C_l|$ is the number of nodes in C_l. Let C_{l-1} denote the immediate lower cluster for node n. Node n then creates exactly $\lfloor \log(|C_l|/|C_{l-1}|) \rfloor$ new links within C_l, thus retaining the $\lfloor \log |C_{l-1}| \rfloor$ long-distance links n uses to route within C_{l-1}. Consequently, n routes within every cluster C_l as it would do in a typical $|C_l|$-node Symphony overlay. Further, Geophony maintains K XOR links for the top $cl = K$ cluster levels, for some little constant K. The XOR metric distance between two IDs x and y is $d(x, y) = \Sigma_{i=0}^{b}|x_i - y_i| \cdot 2^i$. The idea is to apply the XOR metric on *clusterIds* to efficiently route between clusters. These links will contact to nodes within clusters at a distance $d \in [2^i..2^{i+1})$ away, for $0 \leq i < K$. To do so, it requires no global information about the hierarchical structure. It suffices for each node p to know its own path in the hierarchy in order to compute *clusterIds* for all p's clusters. Because in this scheme most significant bits in *clusterId* are the rightmost bits, we need to reverse *clusterIds* and key's suffix part in order to evaluate correctly the XOR metric distance.

To make Symphony compatible with Cyclone, all that is necessary to do is to transform the Symphony real unitary keyspace $[0, 1)$ into keyspace $[0, 2^b)$ and to make the probability function that nodes use to create long-distance links *discrete*.

4.1.1 Routing

Conventional routing in Geophony is identical to routing in Symphony, namely, *absolute greedy routing*, but operating in loops. In the first loop, a node that wishes to route a query for a key *k*, it initially routes the message to the closest node *p* of *k* within its lowest cluster. In the second loop, *p* switches to the next higher cluster and continues routing on that cluster. The same operation is repeated in each layer until the node responsible for *k* is reached. As the routing procedure goes up, more and more peers are included and the message is closer to the destination. At the last loop, routing is executed on the global cluster, which includes all peers in the system. It can be verified without much difficulty that Geophony achieves *logarithmic* routing when each node has degree $O(\log N)$ (as in Symphony). Fig. 2 depicts an example.

Note that this conventional routing enables to look up the responsible node of a query, but it is unable to locate a sibling cluster, in order to perform there some operation (e.g. a geographical query). *XOR routing* appears to deal with this challenge. XOR routing employs only the *K* XOR links and is similar to that in [16]. Node *n* routes greedily a query *q* to its *i*-th XOR node link *p* that minimizes the XOR distance between query *q* and node *p*, from all the XOR link set. This process is repeated until an arbitrary node *m* and query *q* share exactly the *K* rightmost bits. This means that the query has reached successfully the destination cluster. It is easy to demonstrate that with an amount of CL clusters, XOR routing reaches the destination cluster in $O(\log CL) \leq K$ node hops. Finally, conventional routing is employed to reach exactly the node responsible of query *q* within the destination cluster (see Fig. 3).
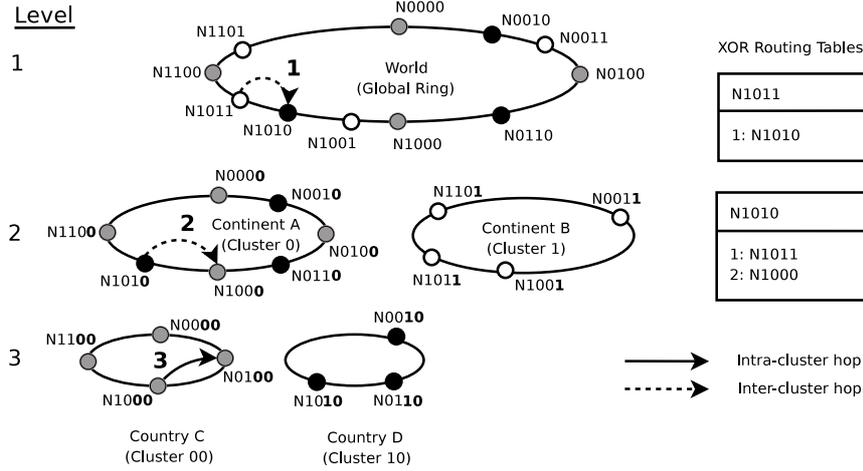
Figure 3. Node $N1011$ **needs to locate** Country C **(codified as** 00**) where to look for a location-based service (codified as** 01**). Steps 1 and 2 are part of (top-down)** XOR routing **in order to locate destination cluster. Step 3 is the part of** conventional routing **to locate the destination node (owner of key** $K0100$**).**

4.1.2 Properties

As noted in [24], a direct consequence of using suffixes as *clusterIds* is that Cyclone's performance is not affected by imbalances on clusters population, since Cyclone distributes uniformly the nodes within each cluster. Note that routing efficiency is based on the assumption that nodes are uniformly distributed along $[0, 2^b)$. To see why, consider that nodes in a lowest cluster C use the cl righmost bits of their node IDs to represent C *clusterId*. Then, it is easy to see that by assigning the $b - cl$ leftmost bits of their node IDs *uniformly at random*, these nodes become evenly distributed within C. Thus, it results in routing load balancing over the whole set of nodes, what is one of the important features of our approach.

Consistent with the previous discussion, Geophony provides another two key benefits. On the one hand, it makes sure that the path from a node to another never leaves the lowest cluster which contains both nodes, property known as *path locality*. On the other hand, it allows to store (resp. retrieve) information into (resp. from) a specific cluster representing a real geographical area, property known as *content locality*. Path locality provides *fault isolation and security*, since interactions between nodes within a cluster are not interfered with by node failures outside the cluster. Also, it turns out to be that path locality is strictly related to caching. For effective caching, one can easily exploit the fact that queries for the same key always exit a cluster through the same node, which we call *exit point*. (Notice that this node is the closest neighbour to the key within that cluster). Then, by caching the answer for the key at each exit point on the path to the query destination, we will reduce the number of routing hops to get a response. Further, this scheme benefits to nicely enjoy additional operations such as aggregation (i.e. known as

11

SUM, *MAX*, *MIN*, *AVG* and *COUNT*). For example, using *MAX* operation one could list the *top ten* restaurants in an arbitrary country.

In Section 5 we will explain and validate how we achieve routing and data load balancing taking advantage of Geophony path locality and caching at exit points. In the rest of the paper we will refer to Geophony, but other homogeneous hierarchical DHT instances are possible. Let us now explain in detail the suffix-based assigment scheme over Geophony *clusterId*.

4.2 Location-based node IDs over Geophony

We propose to divide the node ID into two fragments. One of them is the *clusterId*, that identifies a cluster, and the other is the $nodeId$, that identifies a node within a cluster. Unlike the prefix-based $clusterId$ assignment performed in [35], we define a suffix-based $clusterId$ that nicely fits the Cyclone's hierarchy. Let us to define $|nodeId|$ and $|clusterId|$ as the precisions of these parts measured in number of bits, respectively.

Now, how $nodeId$ and $clusterId$ are defined for every node in the system? The $nodeId$ is drawn uniformly at random from the keyspace $[0, 2^{b-cl})$ thus guaranteeing a uniform distribution for load within every cluster. The $clusterId$ gets a value representing its *geographical location*, given by the administrator in the joining process. We fix $|clusterId|$ to 40 bits because this length provides a good level of precision. For having a practical view, a precision of 40 bits would imply to address a geographical area per cluster of approximately $463.93m^2$ of the total Earth's surface. Ruling out the total water surface, we could even address an area of $134.54m^2$. The rest of the node ID corresponds to the $nodeId$. If one chooses 160 bits as in Chord (i.e. $|nodeId| = 120$), up to 2^{120} nodes would support every subarea. We believe that these bit precisions are *necessary and sufficient* for the present and a long time. Nevertheless, one can set up greater bit precisions, addessing even a smaller area.

The length of the $clusterId$ is partitioned in 3 geocluster levels, from right to left: *continent* (3 bits), *country* (7 bits) and *region* (30 bits) divisions (see Fig. 4). The region level guarantees a correct geographic precision. Instead of assigning random codes to all these divisions, we employ SFCs for guaranteeing a certain geographical contiguity. Therefore, enabling queries for *local information* and even *aggregated information* for all these geoclusters levels. We set the codes for each geocluster level *recursively*. Thus, firstly, we apply the SFC on the $\{latitude, longitude\}$ centroid coordinates to set the code for the continent division (see Fig. 1); secondly the country division code and, lastly, the region code is set accordingly to the specific $\{latitude, longitude\}$ coordinates. For the simplicity of the algorithm, we have employed Z-curve [19] SFC, but other SFCs are also applicable (e.g. [13]). The region code is computed online by each node when joining or inserting data into the system; of course, always accordingly to the specific $\{latitude, longitude\}$ coordinates. It is easy to see that all nodes belonging to different regions but to the same country, will appear merged in the country geocluster. Thus, we fix the number of XOR links K to be 10 (3+7), the number of necessary links to locate a country geocluster. Nonetheless, if a Geophony instance sets $K = 40$, it will be able to address even a region geocluster.

Note that this ID construction is static (will not variate over the time) and independent of the peer-to-peer substrate instance, thus promoting a DHT-generic approach. Additionally, the $clusterId$ helps to group nodes by
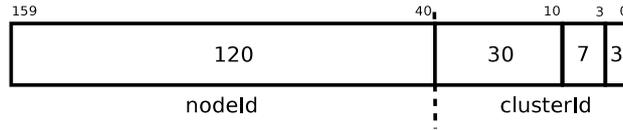
Figure 4. 160-bit node ID structure example: 120-bit *nodeId*, 40-bit *clusterId* (3 bits continent, 7 bits country, 30 bits region).

geographic affinity. Recall that our methodology is also applicable to other data domains, like business organization or knowledge fields. As it occurs with continent, country and region granularities, the only necessary thing is to embed the most general concepts into the righmost bits of the *clusterId*, and successively the other more specific concepts just into the following rightmost free bits (see Fig. 4). This also promotes a domain-generic solution.

5 Achieving Routing and Data Load Balancing

As our methodology is DHT-generic, we do not evaluate how good is the specific DHT routing. Instead, we believe that we must analyse how the resulting homogeneous hierarchical DHT can achieve routing and data load balancing and, moreover, given that information is not uniformly spread over the world.

In a ring-based overlay, the arc length defines both the incoming connectivity degree and the keyspace segment responsible for every node. Because nodes appear uniformly distributed at random within the leaf clusters, Geophony does not impose any responsibility unbalancing on the arc length and, in consequence, enables *routing and data load balancing*. Nevertheless, Geophony suffers from hotspots for skewed datasets in the same way that Symphony or other DHTs like Chord.

To almost mitigate the effects of this situation, *caching and replication techniques* appear in structured peer-to-peer systems. Geophony defines inherently a caching technique based on the query path. This path travels always across cluster *exit point* nodes. Thus, caching requested information on these nodes will mitigate without doubt the query load of the responsible node. Moreover, within every cluster one can define a replication algorithm and, therefore, mitigate even more the incoming traffic of the responsible node. With both data caching and replication techniques, we believe that load balancing in the overall system is successfully addressed even for skewed datasets.
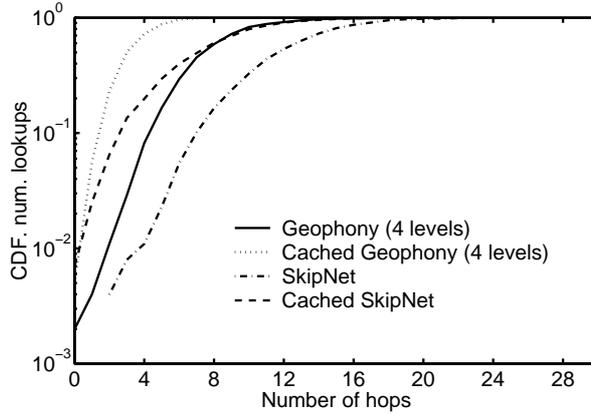
5.1 Routing and Data Load Balancing Evaluation

We present here some simulation results to validate and illustrate the contributions we achieve with our methodology. As we have seen, Skip Graphs/Nets have been having a lot of interest lastly for their properties, capabilities and above all for providing nicely range queries. For this reason, we compare Geophony against SkipNet.

To make a fair comparison, we assume that in both systems each node maintains $O(\log N)$ neighbours as we discussed above (in this scenario the other K XOR links are useless). We choose $b = 24$, so node IDs are binary strings of 24 bits. Also, we vary the number of levels from 1 (Symphony) to 4, with $|clusterIds|$ of 0, 3, 5 and 7 bits, respectively. We also assume a Normal distribution assignment of each node to a leaf geocluster,

13
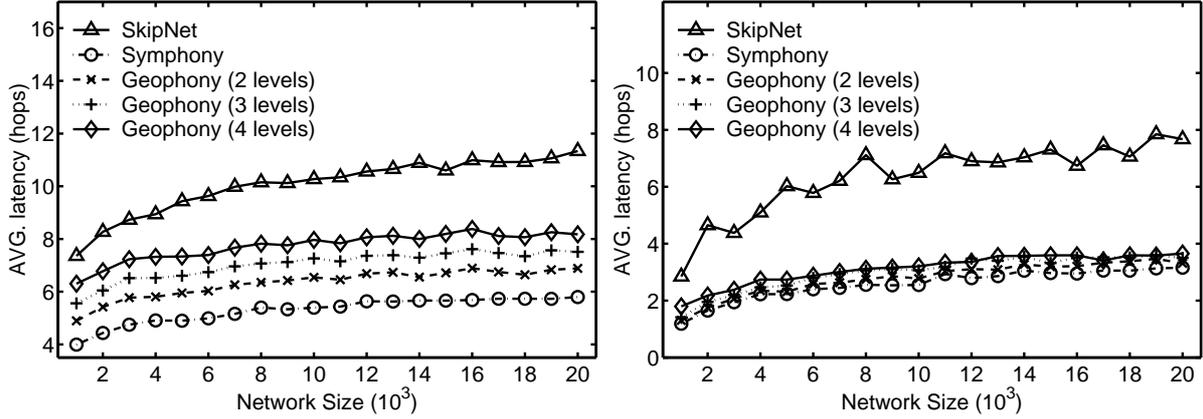
(a) Routing hops without caching.

(b) Routing hops with caching.

(c) Caching effect evaluation.

Figure 5. Routing load balancing. Average of 10K queries from randomly selected nodes.

with $\sigma = 0.125$ and a distributed uniformly at random μ. The reason why we use such skewed distribution is to emphasize the fact that Geophony, similar to that happens in SkipNet, is insensitive to the local distribution of nodes in each cluster. We vary the number of nodes between 1K and 20K, and, for each network size, we run at least ten differently seeded experiments, consisting of *10K random requests*, unless otherwise noted.

Let us first evaluate numerically the average latency (in terms of number of routing hops) of Geophony vs. Skipnet (Fig. 5(a)). Note that a lower average latency means lower network delays experienced by a end user that frequently performs exact match queries. To that effect, in Fig. 5(a) we plot the routing latency averaged over all nodes and all requests. Although the number of links per node is $O(\log N)$ for both geometries, the figure shows that Geophony provides a lower average latency than SkipNet, irrespective of the number of levels in the hierarchy. However, it is important to note here that the average latency increases *slightly* when the number of levels in the hierarchy increases. We note, besides, that this increase is at most 2 hops. The reason for this increase lies in Geophony hierarchical routing. While in Symphony every node has all links available to route a query, Geophony

forces a query to cross the exit points at each level, inducing a path that is not always the optimal route (notice that a node does not have available all links until level 1).

We next evaluate the performance of both systems assuming that all nodes are able to cache answers (Fig. 5(b), 5(c) and 6). We assume that both geometries use *path caching* that consists in caching the answer on all nodes through which the query is routed. We use path caching since it is considered the most deployed caching technique for overlay networks [8]. Although at first glance it may seem inappropriate to use caching as metric to compare both systems, we argue that it is important to measure the potential reduction in latency a system can experience in the face of reiterative queries. We point out that this evaluation is somewhat complementary to that in Fig. 5(a), but remarking the fact that a hierarchical substrate is ideal to handle geographical information, beyond scalable and resilient. Our motivation stems from the observation that caching, and more generally, content delivery networks, are one of the most deployed applications of network overlays.

Figure 6. Number of caching copies, fixing lookups to an average of 4 hops from all nodes within the network.

Firstly, we perform the following two simulations. In the first simulation, we evaluate the latency reduction that geometries experience when the maximum number of answers to be cached for a given query is set to 128 (Fig. 5(b)). Geophony, which takes advantage of exit points, experiences a reduction in latency greater to that in SkipNet, which stores answers at nodes that might not be on the route to the destination. We can see also from Fig. 5(c) that, on the existence of caching in both systems, Geophony achieves clearly a reduction of the number of hops per query (almost the 95% in 4 hops).

In the second simulation, taking the opposite view, we investigate how many caching copies are required to store an answer A, so that the average latency to access A does not exceed a total amount of 4 hops (Fig. 6). To that effect, we assume that all nodes perform the same query. For the sake of clarity, in this plot we only include the results corresponding to a 4-level Geophony, which is the worst setup for this experiment (recall that the average latency increases with the hierarchy depth). As expected, SkipNet requires to cache more answers than Geophony to maintain an expected latency of just 4 routing hops. Also, notice that the number of cached answers remains nearly constant, irrespective of the number of nodes in the system, thus proving the intuition that *with exit points the efficacy of caching is greater*.
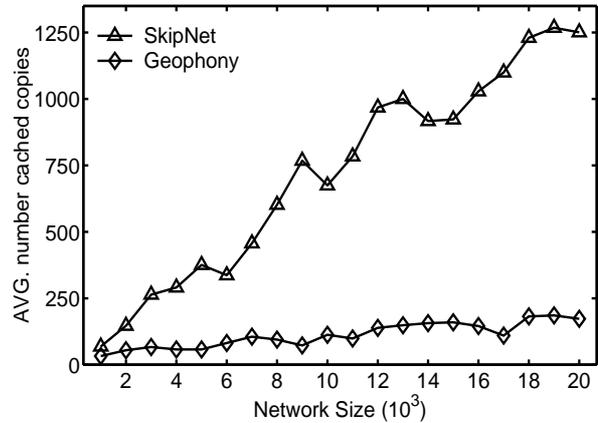
6 Providing higher-level queries.

Our system provides three kind of queries: *exact match queries combining tag and geotag information*, *spatial range queries* over geographical coordinates and a novel proximity abstraction entitled *geocast query*. As explained before, geotags are encoded in the $clusterId$ and tags are stored accordingly in the cluster. This benefits from our homogeneous hierarchical system designed for storing and locating information in specific clusters.

6.1 Exact match queries

Insertion and deletion of information in Geophony combining a geotag (lat,long, geocluster) and a tag (semantic information) simply implies calculating the appropriate node ID. As explained before, the geotag is encoded using SFCs in the suffix, and the tag is encoded in the remaining prefix. As a consequence, the information will be stored in the cluster specified by the geotag (i.e. *clusterId*) and in the node responsible for storing the key (i.e. *nodeId* part) inside that cluster (see Alg. 1).

Algorithm 1 Exact match query: $n.exact\_match(querying\_node, geotag, tag, level)$

/*n is the querying node. By default, $level$ is set to $REGION$. $higher(level)$ returns just the inmediate higher granularity accordingly, i.e. COUNTRY, CONTINENT and eventually WORLD.*/

$key.prefix = $ encode tag

$key.suffix = $ encode $geotag$

$exit\_point = n.conventional\_routing\_send(key, level)$

while $level \neq WORLD$ **do**

  $level = higher(level)$

  $m = exit\_point$

  $exit\_point = m.conventional\_routing\_send(key, level)$

end while

$result\_set = exit\_point.local\_search(tag, level)$

$exit\_point.direct\_send(result\_set, querying\_node)$

We see that with simple *put* and *get* operations we can efficiently store and retrieve *multidimensional data combining both tags and geotags*. Furthermore, if we store data in lexicographic order, an OPHF (like in SkipNet), we provide data locality inside each cluster. Again, it is obvious that skewed datasets can imply load balancing problems, but the cluster-based caching techniques presented in the previous section can almost overcome those problems.

Searching for information like *<geotag:lat,long, tag:jazz>* involves a simple lookup operation in the overlay with *optimal logarithmic routing cost*. Additionally, we can also perform queries using *wildcards* like in *<geotag:lat, long, tag:*>* and *<geotag:*, tag:jazz>* at the same cost. Specifically, the first wilcard enables the user to retrieve any sort of information existing in a certain place (geocluster). The second wildcard enables to

retrieve any information related to *jazz* stored at the global ring.

6.2 Spatial range queries

Traditional spatial databases store geographical data using hierarchical structures (R-tree, Kd-tree) by defining Minimum Bounding Rectangles (MBRs). Leaf nodes in the R-tree contain entries of the form $(data, mbr)$ and non-leaf nodes contain entries of the form $(ptr, rect)$, where ptr is a pointer to a child node in the R-tree and $rect$ is the MBR that covers all the MBRs in the child node. Unfortunately, adapting these algorithms to a distributed way is a very complex problem in the big scale.

Like in PlaceLab, our system can be seen also as a trie-based topology that partitions the space and thus permits to have implicit knowledge about key locations in the hierarchy. This clearly fits the DHT lookup mechanisms and it avoids the fragility of distributed tree topologies.

The spatial range query, matching data within a rectangular region defined by $(latMin, longMin)$ and $(latMax, longMax)$, is defined as follows (see Alg. 2). We must calculate the *linearized suffix* that minimally encompasses the entire query zone. Using Z-curve, we first obtain the longest common prefix of minimum and maximum ranges $(zMin, zMax)$ for this query. Then we invert this prefix and we encode it in the *clusterId*. Now, we can use this suffix to traverse the tree until we reach the responsible node(s) of the specified bounding rectangle, similarly in essence to that of [9]. In this situation, a total of $O(\log CL + \log |(zMin, zMax)|)$ hops are needed to deliver the query to the responsible node(s), where CL is the total number of clusters and $|(zMin, zMax)|$ is the number of nodes into the search area of the responsible cluster C_i. Note the upper bound $|(zMin, zMax)| \leq |C_i|$, when search area overlaps the whole geocluster area. We forward the query through Geophony XOR routing to locate the target cluster, employing $O(\log CL)$ number of hops to realize such an operation. Afterwards, conventional routing delivers the query to the responsible node(s) in that cluster with $O(\log |(zMin, zMax)|)$ hops. Fig. 3 depicts an example of this combination of routing schemes.

We directly take advantage of suffixes (i.e. *clusterId*) in Geophony to efficiently locate *multidimensional data* in specific clusters. In addition, we can combine these spatial range queries with *semantic keywords* (tags) to further filter search information. For example, one can search using Google Local all companies of a certain type (tag:restaurant) in a specific geographic bounding rectangle. In our case, *we offer the same functionality but using a completely decentralized, scalable architecture*.

Algorithm 2 Spatial range query: $n.spatial\_range\_query(querying\_node, (zMin, zMax), tag, level)$

/* n is the querying node. */

$clusterId =$ encode in suffix $common\_prefix(zMin, zMax)$

$node = n.XOR\_routing\_send(clusterId, level)$

$node.process\_spatial\_rq(querying\_node, (zMin, zMax), tag, level)$

Algorithm 3 Spatial range query (cont'd): $n.process\_spatical\_rq(querying\_node, (zMin, zMax), tag, level)$

/ Parallel range queries. n is a node responsible of a part (or the whole range) of the spatial range query. */*

$node\_set = \{node \in n.routing\_table : node.location \in (zMin, zMax)\}$

for all $node$ in $node\_set$ **do**

    $(nzMin, nzMax) =$ area of $(zMin, zMax)$ owned by $node$ within $node\_set$

    $node.process\_spatial\_rq(n, (nzMin, nzMax), tag, level)$

end for

$result\_set = n.local\_search(tag, level) \cup node\_set.result\_sets$

$n.direct\_send(result\_set, querying\_node)$

6.3   Geocast queries

A geocast query is a new primitive which permits a user to recover information associated with an arbitrary tag (in a similar way that Scribe's anycast primitive does) in all users' geoclusters very efficiently: starting from users' region, continuing on users' country and so on until the global owner of the tag is found.

For example, consider a Spanish SUN researcher that wishes to retrieve the information about the projects in which it participates at all scales, that is, the SUN projects that are being developed in Spain, the European SUN projects in which it participates, and finally the intercontinental SUN projects. Using a geocast query for the tag SUN, our system guarantees that all the above information can be recovered within $O(\log N)$ routing hops, in stark contrast to a conventional DHT, in which typically four queries of $O(\log N)$ hops each would be required. Informally, one query for each pair *<geotag:region, tag:SUN>*, *<geotag:country, tag:SUN>*, *<geotag:continent, tag:SUN>* and *<geotag:world, tag:SUN>*.


Algorithm 4 Geocast: $n.geocast(querying\_node, tag, level)$

*/*First call is $querying\_node.geocast(querying\_node, \{tag\}, REGION)$ and the call to $higher(level)$ returns just the inmediate higher granularity accordingly, i.e. COUNTRY, CONTINENT and eventually WORLD.*/*

$exit\_point = n.conventional\_routing\_send(tag, level)$

$result\_set = exit\_point.local\_search(tag, level)$

$exit\_point.direct\_send(result\_set, querying\_node)$

if $level \neq WORLD$ **then**

    $exit\_point.geocast(querying\_node, tag, higher(level))$

end if

The idea is to iterate the hierarchy (see Alg. 4), routing a query for tag T as usual (as in an exact match query), starting from the lowest geocluster in which the querying node lives, thus taking benefit from the hierarchical greedy routing just as described above. Once the node p_T responsible for T in the regional geocluster is found (i.e. the so-called exit point for tag T in this cluster), p_T returns the result for the pair *<geotag:region, tag:SUN>* if any, and continues routing on the next higher geocluster, known as country geocluster. This operation is repeated
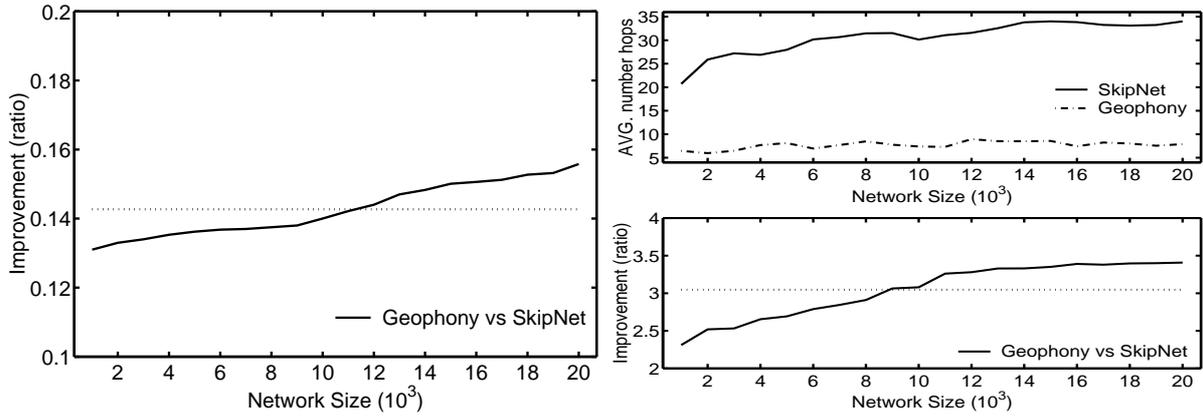
(a) Ratio of improvement on spatial range queries.   (b) Number of hops and improvement on geocast queries.

Figure 7. Higher-level queries. Average of 100K queries.

for each geocluster until the absolute owner for T is reached in the world geocluster. Fig. 2 depicts an example, having *N1100* as the querying node, *N0000* as country exit point, *N0010* as continent exit point and *N0011* as the absolute owner node. Further, after every step an answer to node *N1100* is sent. Thus, we view our geocast abstraction as typical $get()$ operation, which is optimal in number of routing hops. In particular, $O(\log N)$ routing hops on average. To the best of our knowledge, we do not know of any system that offers such functionality as efficiently as our approach. Notice that our geocast queries differ from those evaluated in [31]: geocasting in Mobile Ad hoc Networks (MANETs) means to multicast a message to all nodes that lie in a certain geographical area.

In addition, our abstraction provides other useful utilities. For example, we can insert conditional predicates into a geocast query either to filter information at exit points before sending it to the querying node or to stop the query when some condition is met, thus saving bandwidth and reducing network delays. We call this variation *anygeocast* query. Other variations are possible, but for want of space, we have not included them in this paper. Our experimental results show that Geophony clearly outperforms SkipNet in this aspect. At least for us, this abstraction has yielded new exploitable insights.

6.4 Spatial and Geocast queries evaluation

We evaluate spatial range queries and our novel geocast query on Geophony against SkipNet. On the one hand, as SkipNet is a data-centric Skip Graph-based overlay, where range queries are performed efficiently, we foresee that both systems will perform in a similar operation cost on spatial range queries. On the other hand, we foresee that Geophony will clearly outperform to SkipNet in the geocast evaluation.

We have simulated the same scenario as explained before, having from 1K- to 20K-node networks, performing a total amount of 100K queries for each test. We employ the same mapping mechanism to both systems, but in SkipNet it appears as a prefix-based mapping, as we have announced in Section 2, in order to take advantage of the SkipNet routing. We evaluate the performance of both kind of queries by the average latency.

19

As SkipNet was designed to enable nicely range queries with a logaritmic cost, both systems result nearly equivalent in spatial range queries, as depicts the Fig. 7(a). Moreover, by means of SkipNet's double routing (by nameID and numbericID), SkipNet and Geophony have routing and data load balancing. Fig. 7(b) shows the geocast evaluation. Here, instead, Geophony improves nearly 3 times in average over SkipNet. It is easy to see that in Geophony, a geocast fits nicely the hierarchical overlay structure, but SkipNet, a flat data-centric DHT, is penalized by performing as many queries as geographical levels appear embedded into the node ID (four in our case: region, country, continent and world).

7 Conclusions and future work

In this paper we have introduced a novel DHT-generic methodology to support geographical queries onto existing DHTs. Using the Cyclone algorithm, we have created Geophony, a hierarchical version of Symphony. We propose a novel suffix-based location node ID assignment in order to map geographical areas (i.e. continent, country and region) and coordinates (using SFCs) to the suffix node ID (i.e. *clusterId*).

On the contrary to what could be initially expected, our approach provides data locality without sacrificing the overall routing and data load balancing properties for skewed node IDs. Furthermore our system supports spatial range queries combining location information (geotags) and semantic keywords (tags). Besides, we also have presented the geocast search abstraction, enabling efficient local incremental queries over geoclusters.

We have provided clear validation results that demonstrate that our approach outperforms flat data-centric overlays (Skipnet) in data load balancing and geocast queries. This is a consequence of the hierarchical clustered architecture of our model. To conclude, we also outline that GTAG is an essential component of the open source SocialDNS project [2].

8 Acknowledgments

References

[1] Gnutella. `http://www.gnutelliums.com`.

[2] SocialDNS. `http://www.socialdns.net`.

[3] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *SIGMOD Record*, 32(3), September 2003.

[4] F. Araujo and L. Rodrigues. GeoPeer: A location-aware peer-to-peer system. Technical Report DI/FCUL TR 03–3, University of Lisbon, December 2003.

[5] J. Aspnes and G. Shah. Skip graphs. In *Proc. 14th ACM-SIAM SDA*, pages 384–393, January 2003.

[6] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. Lamarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. In *Proc. SIGCOMM '05*, volume 35, pages 97–108, New York, NY, USA, October 2005. ACM Press.

[7] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. In *Proc. IEEE P2P'05*, pages 57–66, Washington, DC, USA, August 2005. IEEE Computer Society.

[8] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. HotOS-VIII*, Schloss Elmau, Germany, May 2001. IEEECompSoc.

[9] S. El-Ansary, L. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *Proc. IPTPS'03*, 2003.

[10] L. Garces-Erice, K. W. Ross, E. W. Biersack, P. A. Felber, and G. Urvoy-Keller. Topology-centric look-up service. In *Proc. of COST264 Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany, 2003.

[11] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVÉ: Anonymous Location-based Queries in Distributed Mobile Systems. In *Proc. WWW'07*, pages 371–380, New York, NY, USA, May 2007. ACM Press.

[12] N. Harvey *et al.* SkipNet: A scalable overlay network with practical locality properties. In *Proc. USITS'03*, Seattle, WA, USA, March 2003.

[13] D. Hilbert. Ueber stetige abbildung einer linie auf ein flashenstuck. Mathematishe annalen, 1891.

[14] T. Kawakami, S. Takeuchi, Y. Teranishi, K. Harumoto, and S. Shimojo. A p2p-based mechanism for managing location-dependent contents in ubiquitous environments. In *Proc. SAINTW'07*, page 54, January 2007.

[15] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. USITS'03*, Seattle, WA, USA, March 2003.

[16] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on XOR metric. In *eProc. IPTPS'02*, March 2002.

[17] A. Mondal, Y. Lifu, and M. Kitsuregawa. *P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments*. Springer-Verlag, 2004.

[18] S. Oechsner, T. Hobfeld, K. Tutschku, F.-U. Andersen, and L. Caviglione. Using Kademlia for the Configuration of B3G Radio Access Nodes. In *Proc. PERCOMW '06*, page 141, Washington, DC, USA, 3 2006. IEEE Computer Society.

[19] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proc. PODS'84*, pages 181–190, New York, NY, USA, 1984.

[20] S. Ramabhadran *et al.* Prefix Hash Tree: An indexing data structure over distributed hash tables. In *Proc. PODC'04*, pages 368–368, New York, NY, USA, 2004.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. SIGCOMM '01*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[22] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, volume 2218, pages 329–350, November 2001.

[23] H. Sagan. Space-Filling Curves. In *Springer-Verlag*, 1994.

[24] M. Sanchez Artigas, P. Gomez Lopez, J. Pujol Ahullo, and A. F. Gomez Skarmeta. Cyclone: A Novel Design Schema for Hierarchical DHTs. In *Proc. IEEE P2P'05*, pages 49–56, Washington, DC, USA, 2005.

[25] B. K. Shrivastava, G. Khataniar, and D. Goswami. Performance Enhancement in Hierarchical Peer-to-Peer Systems. In *Proc. COMSWARE'07*, pages 1–7, January 2007.

[26] 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. SIGCOMM '01*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[27] X. Sun. SCAN: A Small-World Structured P2P Overlay for Multi-Dimensional Queries. In *Proc. WWW'07*, pages 1191–1192, New York, NY, USA, May 2007. ACM Press.

[28] M. Xiujun, L. Gang, X. Kunqing, and S. Meng. A peer-to-peer approach to Geospatial Web Services discovery. In *Proc. INFOSCALE'06*, page 53, NY, USA, 2006. ACM Press.

[29] Z. Xu and Z. Zhang. Building low-maintenance expressways for P2P systems. Technical Report HPL-2002-41, HP, 2002.

[30] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. ICDE'03*, pages 49–60, 5-8 March 2003.

[31] P. Yao, E. Krohne, and T. Camp. Performance comparison of geocast routing protocols for a MANET. In *Proc. ICCCN'04*, pages 213–220, October 2004.

[32] C. Zhang, A. Krishnamurthy, and R. Y. Wang. SkipIndex: Towards a scalable peer-to-peer index service for high dimensional data. Technical Report TR-703-04, Princeton University, May 2004.

[33] C. Zhang, A. Krishnamurthy, and R. Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *Proc. IPTPS'05*, volume 3640. Springer, February 2005.

[34] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, Berkeley, CA, USA, April 2001.

[35] S. Zhou, G. R. Ganger, and P. Steenkiste. Location-based node IDs: Enabling explicit locality in DHTs. Technical Report CMU-CS-03-171, Carnegie Mellon University, September 2003.

[36] R. Zimmermann *et al.* Efficient query routing in distributed spatial databases. In *Proc. GIS'04*, pages 176–183, New York, NY, USA, 2004. ACM Press.