Computer Communications 33 (2010) 984–996

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Replica-aware, multi-dimensional range queries in Distributed Hash Tables

Antony Chazapis *, Athanasia Asiki, Georgios Tsoukalas, Dimitrios Tsoumakos, Nectarios Koziris

Computing Systems Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, Greece

ARTICLE INFO

Article history:
Received 7 May 2009
Received in revised form 4 September 2009
Accepted 26 January 2010
Available online 1 February 2010

Keywords:
Peer-to-peer systems
Distributed Hash Tables
Replication
Multi-dimensional data
Range queries

ABSTRACT

In this paper, we present and evaluate a protocol that enables fast and accurate range-query execution in Distributed Hash Tables (DHTs). Range queries are of particular importance when the network is populated with groups or collections of data items, whose respective identifiers are generated in a way that encodes semantic relationships into key distances. Contrary to related work in the same direction, our proposed query engine is aware of data replicas at the DHT level and by grouping related nodes into replica *neighborhoods*, resolves queries with the minimum amount of messaging overhead. Moreover, we suggest pairing respective operations with the core DHT routing mechanics, which allows for reusing existing management and monitoring structures and automatically adapting the query path to the dynamic characteristics of the overlay. We also present an application scenario and the respective deployment details of a prototype implementation in the context of the *Gredia* project.

1. Introduction

Peer-to-peer computing has been widely accepted as a robust, easily deployable paradigm on which fully distributed, scalable applications can be built. Respective protocols focus on defining the transactions between autonomous network endpoints that collectively form an extensible and resilient substrate – a *network overlay*, which can operate unsupervised and dynamically adjust itself to unadvertised node arrivals and departures, or sudden network black-outs. *Structured* systems, also commonly known as *Distributed Hash Tables* (DHTs), represent a major class of peer-to-peer arrangements that abstract the aggregate network as a key-value based indexing and storage facility. All available designs, share the common concept of *hashing* both peers and data values to identifiers that represent slots of a pre-defined virtual fabric. In the identifier space, each node takes on the responsibility of storing values and managing operations that refer to data with IDs "close" to its own. DHT implementations are designated by the shape of the identifier space used and consequently the corresponding distance function [1]. Kademlia [2] uses an XOR metric, which implies placement at the leaves of a binary tree, while Chord [3] organizes all IDs clockwise around a circle.

DHTs pose as perfect candidates for replacing "traditional" centralized or layered global-scale databases of information that can be easily transformed to a key-value form (i.e. DNS records [4]). Their simple *put/get* interface allows the direct application of traditional programming models to a global scale. Nevertheless, their applicability is limited, due to the lack of inherent support for processing related subsets, or *ranges* of data identifiers. Expressing *range queries* in multiple simpler primitive lookups for all intermediate IDs is not scalable. Ideally, such operations should be internally handled by the overlay, while both respecting and sustaining the scalability and fault-tolerance mechanics that have made DHTs a prominent platform for supporting distributed applications.

This also reflects a common focus of many recent research initiatives, that propose *advanced* or *complex* DHT lookup functions – functions which exploit semantic relationships between data items, including similar or nearest neighbor searching or processing aggregate sets of closely related items. As group operations require data ordering, the hash function for placing values into the identifier space is usually replaced with a locality-preserving mapping scheme. When the data to be distributed is already tied to one or more indices, such a scheme can transform their numeric representations to a single identifier, while maintaining their original organizational and clustering properties. Example mapping functions for *multi-dimensional* values may be based on the *Z* or Hilbert *space-filling curves* (SFCs).

Motivated to implement a DHT capable of performing range queries on multi-dimensional data, we have investigated numerous systems presented in related work and found that most exhibit shortcomings that originate from their two-level architecture, choice of range-query resolution algorithms, or both. Using separate layers for query handling and routing complicates operations,

* Corresponding author. Tel.: +30 210 7722867; fax: +30 210 7721292.
E-mail addresses: chazapis@cslab.ece.ntua.gr (A. Chazapis), nasia@cslab.ece.ntua.gr (A. Asiki), gtsouk@cslab.ece.ntua.gr (G. Tsoukalas), dtsouma@cslab.ece.ntua.gr (D. Tsoumakos), nkoziris@cslab.ece.ntua.gr (N. Koziris).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.01.024

as each layer's data structures must be properly maintained and synchronized to propagate new values or node membership changes. Additionally, the process of translating multi-dimensional ranges to DHT identifier segments of the routing layer may be a very demanding processing task. A common trend to avoid the processing load is to distribute it among the nodes of the DHT, via a *query refinement* process. Such systems match the recursive translation algorithm to a tree-like ordering of peers. This may incur heavy loads at the nodes placed at the higher levels of the overlay's structure and by design does not solve the problem of computing the ID segments corresponding to each multi-dimensional range request. Furthermore, to our knowledge, no range-query capable system is compatible with DHT protocols that inherently handle data replication. While pioneering DHTs could only ensure data reliability via well-defined node disconnect procedures, modern overlays self-replicate each stored value to a dynamic group of close peers. Related multi-dimensional query platforms usually build upon a simple DHT substrate, optimized for routing, and delegate data management and replication to a separate layer. However, when such copies are already available in the overlay, the query engine should account for them accordingly.

In this paper, we present the necessary protocol enhancements that enable a DHT to handle range queries. We do not propose a new peer-to-peer architecture, but rather exploit existing internal structures to introduce a novel routing strategy for looking-up ranges of identifiers. In contrast to other designs, we do not employ a recursive multi-dimensional range to segment conversion, but apply a serial, segment-to-segment hopping algorithm that in addition to its improved efficiency allows the query process to be easily distributed and parallelized. Analogous query resolution mechanisms have been extensively used in database-related contexts, but not in DHTs.

Moreover, by working directly at the DHT level, we achieve several benefits. First, range queries self-manage the path taken throughout the network and avoid visiting nodes which would otherwise reply with duplicate values – *replica awareness* has been an important challenge throughout the design process. To this end, we introduce the notion of *peer neighborhoods*, as groups of peers that store the same replica set. Neighborhoods interact directly with the query engine and serve as a tool to help us estimate and measure the performance characteristics of the network, as well as balance the load over peers of the same group. Inherent DHT replication is not considered an obstacle, but is exploited in favor of faster query resolution. Second, the resulting DHT requires no extra management to support range queries. Since we do not introduce any new routing tables or other internal structures, there is no need for additional continuous or periodic maintenance. Therefore, we stress the importance of embedding range-query support at the core of the protocol. Implementing advanced functions at the lowest level, allows us to exploit the underlying DHT mechanics and let the query automatically adjust its execution depending on the ongoing node and data-placement relationships in the overlay. This degree of integration is also one of the main contributions of this work.

The remainder of this paper is organized as follows: In the next section we discuss on related work in the area. Section 3 introduces our protocol starting with the single-dimensional case, before moving on to multiple dimensions. In Section 4 we present both a formal and an empirical approach to estimate the performance of the proposed algorithm, show evaluation results and elaborate on its behavior under different overlay runtime parameters and data distributions. We then describe the scenario that has motivated us to deal with the problem, the final implementation and deployment details and conclude. A working prototype has been incorporated in the EU-funded *Gredia* project [5] infrastructure to provide an efficient and scalable solution for metadata search.

2. Related work

DHT overlays [2,3,6–8] have been established as an effective solution for data placement and *exact match* query routing in scalable network infrastructures. In respective protocols, values can only be located if the their exact keys are known in advance, while keys – randomly generated by a cryptographic hash function – do not contain any semantic information about the content. As applications demand more complex types of queries, capable of exploiting inter-relationships between data, research efforts have focused on developing corresponding algorithms and mechanisms. Resolving range queries in a peer-to-peer network requires the ordering of values, which comes in conflict with the random assignment of items to nodes. The methodology used to solve this problem, provides a basic categorization of proposed systems encountered in the bibliography:

- Overlays that rely on an existing DHT protocol and either modify/replace the hashing function or add additional indexing structures on top of the DHT.
- Overlays that distribute the dataset to peers, while not directly utilizing any existing DHT protocol.

In the rest of the section, we elaborate on related work from the field of structured peer-to-peer overlays, starting from systems supporting range queries for data items described by a single attribute, before proceeding to platforms that support multi-attribute range operations.

An early approach to support range queries in a DHT overlay is presented by Gupta et al. [9]. The authors store partitions of a relational database in a DHT and enable querying ranges one attribute at a time, by mapping them to identifiers via a special hash function. Nevertheless, hashing ranges is not efficient, as it is possible that similar or overlapping spaces are mapped to different nodes. To avoid flooding the network or following various distributed indices to reach the requested data, they introduce *Locality Sensitive Hashing*, so mappings preserve locality with high probability. However, the probabilistic scheme returns approximate results, whose quality depends on the complexity of the function. Additionally, there are load-balancing issues. Another method of adjusting hashing to enable range queries is described by Andrzejak and Xu [10], where the Hilbert space-filling curve is used for mapping ranges of an attribute's values to parts of a CAN-based overlay. The SFC offers the advantage that nearby attribute ranges are mapped to nearby CAN zones. A query is initially routed to the node responsible for the middle point of the range and then recursively propagated according to three proposed and evaluated 'flooding' strategies.

A common solution for the execution of range queries is the exploitation of trees as additional indexing structures on top of DHT-based, independent routing substrates. Gao and Steenkiste [11] implement a Range Search Tree (RST) on top of a DHT overlay, such as Chord. Each level of the tree corresponds to a different granularity of data partitioning, while content is registered with all or various levels of the RST and the corresponding physical nodes of the DHT. Distributed Segment Trees [12] are similar. P-Trees [13] are also decentralized indexing structures, maintaining parts of semi-independent B+ trees on top of a Chord ring. Their purpose is to help route range queries to the appropriate nodes. Because of complex cross-structure management, the authors note that every query cannot be guaranteed to terminate – for example, if a node crashes. Other efforts target the distribution of *tries*. Tries are a generalization of trees for storing and processing strings in which there is a node for every common prefix. In tries, the actual data is stored in the leaf nodes and thus lookups are resolved by finding the leaf whose label is a prefix of the queried value. The

Prefix Hash Tree (PHT) [14] is such a trie-based structure layered on top of a DHT to enable one-dimensional range queries. A range query is performed by various DHT lookups and the query cost is data dependent. Chawathe et al. [15] adapt PHTs in the 2-dimensional space of a geographic application. The keys indexed by the PHT are generated by a linearization technique based on the *Z*-curve. The *Distributed Lexical Placement Table* (DLPT) [16] is a dynamically constructed trie for indexing, enabling service discovery in grid environments. A trie index, as a semantic overlay on top of a DHT, is also used by the IMAGINE-P2P platform [17]. Queries are forwarded in an index which contains the semantic relationships of data items stored in a Chord ring. In all aforementioned approaches, the underlying DHT serves as an efficient and scalable layer providing primitive operations, on which generic and easily deployable solutions are based. However, the superimposed structures increase complexity and require additional maintenance and management procedures. Moreover, the adaptability of the system to node arrivals and departures decreases and special care should be taken for keeping all indices consistent. Datta et al. incorporate a PHT in the structure of the P-Grid overlay [18] by suitably organizing the routing tables. The P-Grid tree-based topology resembles Kademlia and the sequential querying algorithm is akin to our methodology for the resolution of one dimensional range queries. Nevertheless, the proposed structure is only designed for handling one attribute. The authors consider the fact of using multiple P-Grids, one for each attribute, as a solution for supporting multi-dimensional ranges. However, this would require an increased query cost for searching in all rings, as well as filtering redundant, duplicate results.

Various proposals eliminate the DHT and construct a similar structured overlay that preserves the ordering of stored values. The Skip Graph [19] distributed structure is based on skip lists [20], which is an increasingly sparse set of sorted double-linked lists of keys. Since skip lists are vulnerable to node failures and lack redundancy, a skip graph combines multiple lists at each level. All the nodes participate in the lowest level, while higher levels are used as shortcuts to reach peers at greater distances. Skip Tree Graphs [21] is a complementary architecture, which uses skip trees instead, and supports aggregation queries and multicast/broadcast operations. Bharambe et al. designed Mercury [22], a system consisting of multiple circular overlays. Each overlay corresponds to a different attribute and each node is responsible for a particular range. Nodes are partitioned into groups called attribute hubs and data items are indexed to the corresponding hub for each individual attribute. A query for more than one attribute is considered as a conjunction of predicates and is routed to all relevant hubs. Since the elimination of the hash function may result in non-uniform partitioning of data among nodes, Mercury emphasizes in load balancing, using random sampling to avoid problems with skewed data distributions. Chord$^{\#}$ [23] is also a circular Chord-based overlay, which stores keys in lexicographical order. SONAR [23] is the extension of Chord$^{\#}$ to support range queries over multiple attributes, although in a topology that resembles CAN. Each node includes neighbors for each dimension in its routing table, to enable range-query processing. Defining a new peer ordering, requires developing explicit methods for reestablishing properties inherently supported by a DHT (logarithmic routing, recovery after node arrivals and departures, load balancing etc). Replication is another issue faced orthogonally in these works. The order preserving property requires that replication cannot be natively implemented and another layer is usually added for this reason.

To support the concurrent resolution of ranges referring to more than one attribute, respective systems construct keys based on all available value identifiers. It is considered that all attributes form a multi-dimensional space, which is commonly mapped to one dimension and vice versa using a space-filling curve. Similar methods have been used for indexing multi-dimensional data in database systems [24–26], image processors, etc. An analysis in [27] shows that mappings based on a Hilbert SFC exhibit better clustering properties, namely the locality between objects in the multi-dimensional space is best preserved in the linear space. In structured peer-to-peer systems, the hash function can easily be replaced with an SFC-based mapping. Squid [28] is a Chord-based overlay enhanced with facilities for partial keyword, wildcard and range searches. Data items described by attributes, are assigned an ID by a function that maps corresponding values to a Hilbert index. A range query is resolved with multiple DHT lookups, by recursively searching SFC clusters. At each search step, longer prefixes of candidate clusters are looked up. If a node is responsible for the IDs with the prefix in question, then it queries its local database. Otherwise, it initiates new lookups for more specific subclusters. Despite distributing the calculation of SFC clusters among overlay peers, the process still imposes relatively large computational and messaging costs. Also, the peers responsible of high order ID bits suffer from congestion, since each query is initially forwarded to them. To avoid load imbalance, Squid relies on the fact that the *d*-dimensional keyword space is sparse and so the data items are assigned to peers roughly in the same way. Multi-dimensional indexing via a Hilbert SFC is also employed by the CISS framework [29]. Assuming a hierarchy for each attribute, performing a range query requires the node responsible for the first key of a candidate cluster to be looked up, before forwarding to succeeding peers, until all relevant objects are retrieved. Load balancing is handled either locally, by exchanging items among heavily loaded nodes and their neighbors, or globally by detecting lightly loaded nodes in the system. SCRAP [30], also enables range queries for one or more attributes, by using an SFC to produce single-dimensional indices for values, which in turn are partitioned to nodes. Range queries are resolved by computing the clusters answering the query and routing them in a Skip Graph network. However, the number of calculated ranges may be very large and definitely depends on the refinement level of the SFC curve. The latter is important because it defines the number of bits used for representing each attribute's value. In our case, we completely avoid the computationally intensive method of mapping range queries to key clusters, in order to provide a realistic and scalable solution.

Some multi-dimensional range query capable systems do not use an SFC for replacing the hash function, but define their own data organization scheme. MURK [30] implements a distributed kd-tree in order to partition the data space to rectangles and assign them to participating nodes. Each node is aware of its neighbors and queries are forwarded along these links. However, load balancing remains a tricky point – especially for dynamic workloads. A distributed quadtree is implemented for indexing and querying of spatial data in [31]. The centroid of each quadtree block is hashed and inserted in a Chord overlay. ZNet [32] dynamically partitions space according to *Z*-ordering and uses Skip Graphs as the network overlay. The nodes may be responsible for continuous ranges of *Z*-addresses of different lengths according to the order of the curve for the specific interval. A range query is initially forwarded to the nodes assigned with the shorter prefixes, before being further refined by computing the next recursion of the curve for producing more specific candidate intervals. Armanda [33], which utilizes FissionE [34], a structured overlay composed according to the properties of a Knautz graph, uses algorithms for single and multi-attribute hashing that preserve entire and partial order, based on the graph properties. These algorithms assign data items to peers in such a way that each peer is responsible for a specific interval of an attribute's adjacent values. The search procedure has been adapted to the naming scheme and ranges with common prefixes can be searched. Again, these approaches do not provide functionalities already available from DHTs, or re-

quire complex procedures for synchronizing data structures at different levels.

3. Supporting single and multi-dimensional range queries in a DHT

In the following paragraphs, we describe the proposed range-query protocol in the context of a Kademlia-based routing scheme. First, we elaborate on how the single-dimensional range resolution can be encoded into a series of DHT routing and lookup commands that retrieve respective values while avoiding replicas. Then, we extend into the multi-dimensional scenario, in order to derive a unified range-query protocol. While we discuss on how the *peer neighborhood* concept can also be applied to circular identifier spaces, we use a binary tree structure to present and evaluate our algorithm, due to its unique symmetrical replication characteristics. Our prototype implementation is also based on Kademlia. Accordingly, we employ a *Z*-curve as the SFC for translating multiple indices to DHT identifiers, due to its simplicity, although any mapping function can be used – we have implemented both *Z* and Hilbert curves.

3.1. Looking-up ranges

Looking-up an identifier in a DHT involves hopping from node to node, using the corresponding routing tables, going at each step closer to the specified key. The closest node in the virtual space must store the value, else the key is non-existent. Disregarding replication, performing a range query for all identifiers in-between a given set of Q_{start} and Q_{end}, requires repeating the above steps *for each subsequent node* in the interval. In a DHT, each node may store multiple identifiers in a range, thus range processing should proceed from node to node – not based on identifiers. Given the sparse distribution of keys in a DHT's address space, unless $[Q_{start}, Q_{end}]$ is an extremely small interval, iterating the lookup process over each Q will result in a large number of unsuccessful operations.

Moreover, in overlays that replicate keys to close participants, the algorithm should automatically skip over peers storing replicated values. To avoid wasting time and bandwidth, starting from the node responsible for Q_{start}, the query should proceed to the next node holding different identifiers, etc. until reaching the one closest to Q_{end}. The number of nodes that should be bypassed at each iteration can only be defined when knowing the details of the virtual space that accommodates the participants, as well as the exact replication mechanics.

Kademlia applies the closest-nodes replication scheme via the κ parameter – a key must be stored to at least κ of its closest nodes. Visualizing this in the context of the binary tree structure implied by the overlay's distance function (XOR), a key along with its replicating nodes share positions at the leafs of a common subtree. Consider the following definitions when discussing replica placement in Kademlia: Given an identifier, let *sub-neighborhood* be any subtree containing it, as long as it has a node population of less than κ. We call the *largest* sub-neighborhood of a key its unique *neighborhood*. In a neighborhood's range of identifiers $R_N = [N_{start}, N_{end}]$, any node in R_N stores at least all replicas whose keys fall in R_N. Or: All data keys in R_N are replicated at least to all nodes in R_N. Naturally, more replicas may still be found in adjacent nodes (outside R_N), as each neighborhood's node-size is always less than the replication factor.

In a deployment of less than κ participants, there is only a single group, whose subtree coincides with the whole identifier space. All nodes are neighbors and share copies of all available data. As the node population increases, immediately upon reaching κ, the single neighborhood splits in two – each corresponding to one of the left and right branches of the tree structure. The process repeats each time the number of nodes in a neighborhood reaches the limit set by the replication factor. A simple example for $\kappa = 2$ is presented in Fig. 1. As nodes gradually populate the network, a new neighborhood is formed for every new participant.

The definition of *neighborhood* as a utility is extremely helpful for implementing and evaluating range operations in DHTs, as it imposes a grouping of nodes that can be directly used as hops by the query algorithm. Neighborhood borders are reported by peers when contacting them, as a pointer to how much of the identifier space can be safely ignored at the next iteration. Each peer is well aware of its neighbors, as it has a better knowledge of close identifiers, so it can easily devise and return the current relationship between close routing table entries and replicas of data items.

In Kademlia, neighborhoods are by definition distinct, non-overlapping node groups. The bidirectional nature of XOR, produces a symmetric replication scheme, which allows for picking *any node* from R_N as a valid contact for processing a query referring to all data items in R_N. In circular overlay arrangements, where each key-value pair may be automatically replicated to its f successors – as a variation of Chord's algorithm that places each value to its immediate successor, neighborhoods, in the same sense, cannot be easily defined. Replication ranges overlap. Nevertheless, for an identifier area R_N, containing $\leqslant f$ nodes, all nodes in $[R_{mid}, R_{end}]$, $R_{start} \leqslant R_{mid} \leqslant R_{end}$ share all values with keys in $[R_{start}, R_{mid}]$. Upon contacting a node in $[R_{mid}, R_{end}]$, a range query can continue with one of its f predecessors with ID $< R_{start}$. If replicas are placed to an identifier's $\frac{f}{2}$ successors and $\frac{f}{2}$ predecessors [7], all ranges containing $\leqslant f/2$ nodes are neighborhoods. There is still overlapping, so avoiding duplicate values may require jumping over f adjacent peers at each step.

In Kademlia, nodes organize their contacts into 160 buckets, each representing an exponentially larger part of the 2^{160} id-space as the distance from the node increases. In practice, a peer keeps κ contacts for half the structure that does not include itself (the left or right primary subtree), κ for half of the remaining structure that does not include itself (a second-level subtree), etc. Using this routing table setup, each peer can find its neighboring nodes by traversing the buckets sequentially, from the closest onwards, as long as the total neighbor-count remains within limits.

On the other hand, contacts may be dynamically managed. In related literature, Kademlia designers use a static routing table structure to explain the protocol, but also propose a dynamic implementation where buckets are created only when necessary. Dynamic contact management is reminiscent of the aforementioned neighborhood construction process. Each node has one dynamic bucket, which holds its closest contacts. Each time the

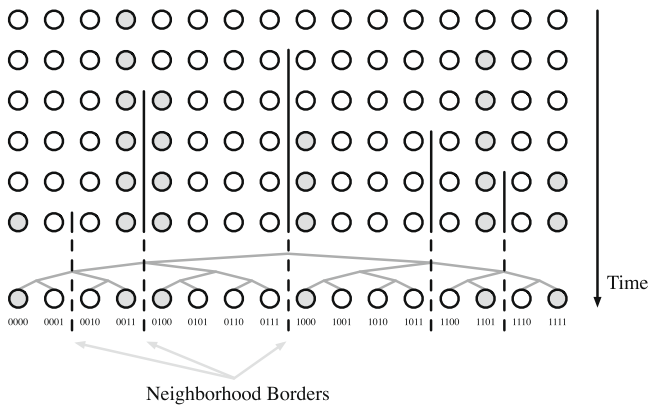

Fig. 1. Example of neighborhood formation for $\kappa = 2$. The identifier space is represented horizontally, as empty slots, filled gradually with nodes (grey dots).

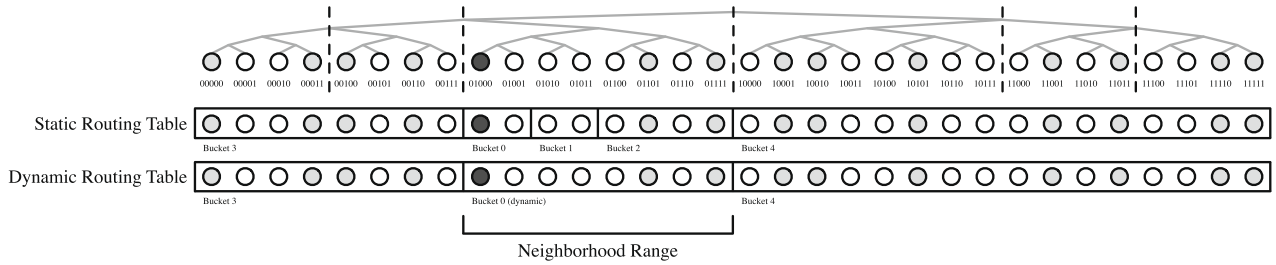

Fig. 2. Routing table implementations for a sample node (dark grey) in a Kademlia overlay with $\kappa = 4$. A dynamic routing table's first bucket contains all neighbors and defines the identifier range of the neighborhood.

dynamic bucket's contact count reaches κ, it spawns a static bucket containing peers that belong to a specific range of identifiers – a distant subtree. New contacts that fall within the range of a static bucket are managed as usual, while all others are added into the dynamic structure. The closest static bucket marks the border between close and distant nodes. It separates the network into the branch containing the node and its less than κ close peers, and all other subtrees. In this type of routing tables, neighborhood information is directly accessible at no extra cost. The first bucket (dynamic) contains all neighboring nodes, while the distance between the first and second buckets reveals the identifier range of the node's neighborhood – it defines the subtree's edges (Fig. 2).

In our implementation we have replaced Kademlia's `FIND_VALUE`/`VALUE` message pair with `FIND_RANGE`/`RANGE`. A `FIND_RANGE` query specifies the range of identifiers that interest the issuer. A `RANGE` reply should contain all stored values in the requested range, plus information about the node's neighborhood – which part of the virtual structure is *covered* by the node. Thus, a range query takes the following steps:

1. Set $Q = Q_{start}$.
2. Lookup the closest nodes to Q.
3. Send a `FIND_RANGE` to the top node in the list.
4. Collect values received in the `RANGE` reply. Add the returned coverage to the total range of covered identifiers.
5. Set Q to the next identifier outside the covered area.
6. If Q is in the lookup range, repeat the process from the second step.

The whole procedure is end-to-end coordinated by one node. This is in agreement with typical DHT mechanics, as it assures that the participant with the incentive to do the operation will also manage its completion. Additionally, it provides a level of fault-tolerance. If, for any Q, a remote node fails to reply, the process can continue with any peer from the same neighborhood. In the example of Fig. 3 ($\kappa = 4$), the node placed at 3, requires three successful `FIND_RANGE` RPCs to complete the query for range [7,20]. Q is initially set to 7, which falls inside the neighborhood [4,7]. Assuming a `RANGE` reply from 6, Q is set to 8, which belongs to the neighborhood [8,15]. Looking up the new Q's closest nodes, results to 8, 13, 15, 0. If node 8 fails to respond, 3 will pick the next peer from the list, namely 13, which should provide an equal data and coverage reply. Faithful to the autonomous nature and the self-management properties of the DHT, the algorithm will automatically skip the appropriate number of peers at each query step, according to pre-defined parameters and the current status of the network.

Moreover, to completely match the original lookup process employed by Kademlia, we allow for an optional asynchronous query step: When getting a `RANGE` reply, the issuer may send parallel `FIND_MORE` requests to all other participants in the peer's coverage in order to retrieve any extra values that may have been missed by previous nodes. `FIND_MORE` RPCs contain the list of data keys already found. In highly dynamic setups, because of the continuously changing associations between nodes and stored values, Kademlia requires that all of a key's κ closest nodes are visited before concluding on whether a data item is indeed not in the network. An unsuccessful point lookup request will not end until all closest nodes found have replied to `FIND_VALUE` requests. We address this issue via `FIND_MORE` messages, but deem their use optional, because the usefulness of the extra communication depends on the deployment environment and the application requirements. It may be known beforehand that the overlay will not experience extreme node churn, or the application may allow for a subset of in-range values to be returned for it to proceed with other tasks.

3.2. Going multi-dimensional

Multi-dimensional values are values that have multiple keys – an identifier for each dimension. In some applications, assigning multiple identifiers for each data item is crucial for pertaining to the semantics of a value. A large amount of related literature is devoted on how to efficiently handle such data: how to store multi-dimensional datasets and how to perform fast and accurate point and range queries in respective collections. A key aspect behind most corresponding proposals is the use of a *space-filling curve* function that maps all indices to a single number. The function is called "space-filling", because it emulates a line that passes continuously through all points of the multi-dimensional space imposed by the values' indices. A key property of an SFC is that it preserves locality in all dimensions during the conversion process, which means that semantically close ranges also fall in neighboring points on the curve. This locality can be exploited either way: In database software, for example, range queries are served using a smaller set of memory page accesses, and the system can automatically deduce related values [35].

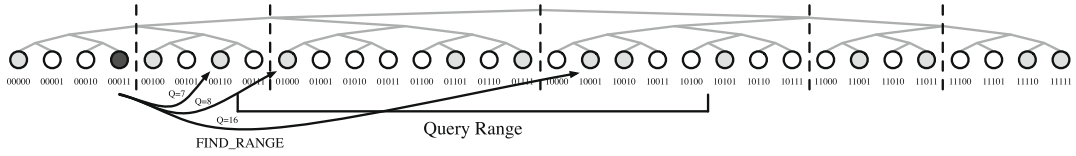

Fig. 3. Performing a range query requires one successful `FIND_RANGE` RPC per neighborhood of query coverage.

There are many SFC algorithms. The simplest mapping can be obtained by concatenating keys to get the aggregate index. While this tactic has some advantages, it preserves locality only in one dimension. Usually, picking an SFC involves balancing the locality-preserving characteristics of the algorithm with its computational complexity. Two widespread SFCs that also scale well in multiple dimensions are the *Z* (Morton order) and the Hilbert curves. Space-walking on the *Z*-curve is accomplished by interlacing the binary digits of each dimension's index. On the other hand, the Hilbert curve follows a self-similar, recursive geometrical pattern, which is constructed in steps, called *orders*. A first order curve, that passes through 2^1 bits in *D* dimensions can be generated by the corresponding Gray code for *D* bits. Higher orders require rotating parts of the graph to avoid intersections. As such, the Hilbert curve is more difficult to implement than the *Z* order, but achieves better locality.

We have extended our DHT implementation to support multi-dimensional data, by incorporating the aforementioned curve generation functions within (both *Z* and Hilbert). When storing or locating multi-indexed values, the key used in the DHT corresponds to the point of the SFC for the given coordinates. Thereby, assuming that each index represents specific characteristics of the value, the SFC will attempt to place semantically close values near each other in the DHT's virtual identifier space. In this setup, range queries become an almost essential feature of the network, in order to take advantage of the relationships encoded in neighboring data items.

Before examining the required steps, consider that, in addition to the one-dimensional case described in the previous section, a range query in multiple dimensions should be converted to a series of corresponding ranges at the level of the SFC. Each space-filling algorithm provides the means of mapping multi-dimensional key ranges to ranges of identifiers on the resulting curve. Nevertheless, the process of computing the later may be extremely time consuming and should be avoided at all costs. A seemingly simple multi-coordinate lookup will commonly produce numerous non-contiguous segments on the curve. Some of these segments may even be single identifiers. It is evident, that computing all segments and issuing a direct or range lookup for each one at the DHT level is not sufficient. Again, the challenge lies in outperforming the naive approach of iteratively going over each part of the overall range, without taking into account that the parts may not necessarily be completely independent.

We propose an approach that exploits the properties of peer *neighborhoods*. In accordance to the one-dimensional protocol, the goal is to traverse each relevant peer neighborhood at most once per range query. At each step, a node should reply with all values that fall within its coverage – the borders of its neighborhood. To jump over to the next node, we use a special SFC function called `find_next(query, id)` that returns the next key, after the id, that is inside the query [25]. The complexity of `find_next()` depends on the SFC type, but in any case it is much more efficient to serve the query from the neighborhood point-of-view, than attacking the problem from the SFC algorithm, even if the requested range covers a large portion of the index space and therefore, requires visiting a large number of neighborhoods.

The process is shown in Fig. 4. The top part illustrates a sample six-bit, two-dimensional space filled with a *Z*-curve. A range query in two dimensions is represented as a bounding box containing all values that should be returned. The equivalent Kademlia network at the bottom of the figure is formed by 13 nodes – their positions in the overlay result in the neighborhood borders indicated by the bold, dashed lines. There is a one-to-one relationship between DHT neighborhoods and respective parts of the *Z*-curve. For example, the first half of the curve corresponds to the left first-level subtree of Kademlia. Thus, neighborhood borders are also shown in the context of the two-dimensional space. The simple query box may cross over four neighborhoods, but produces seven distinct segments at the SFC-level (four of which are single points). However, the range query requires only one visit to each neighborhood and is completed in four steps, instead of seven. Individual query segments that fall in the same neighborhood are processed all together by the same node.

Most of the range query steps previously discussed remain unchanged, except for step 5 which uses the `find_next()` function to get the next *Q* outside the aggregate coverage. The next *Q* may be the next identifier if the previous node's coverage stops in the middle a segment, or a distant key if the coverage overlaps the respective ranges. Also, `FIND_RANGE` messages contain the original range query, as issued in multiple dimensions, so the receiver can

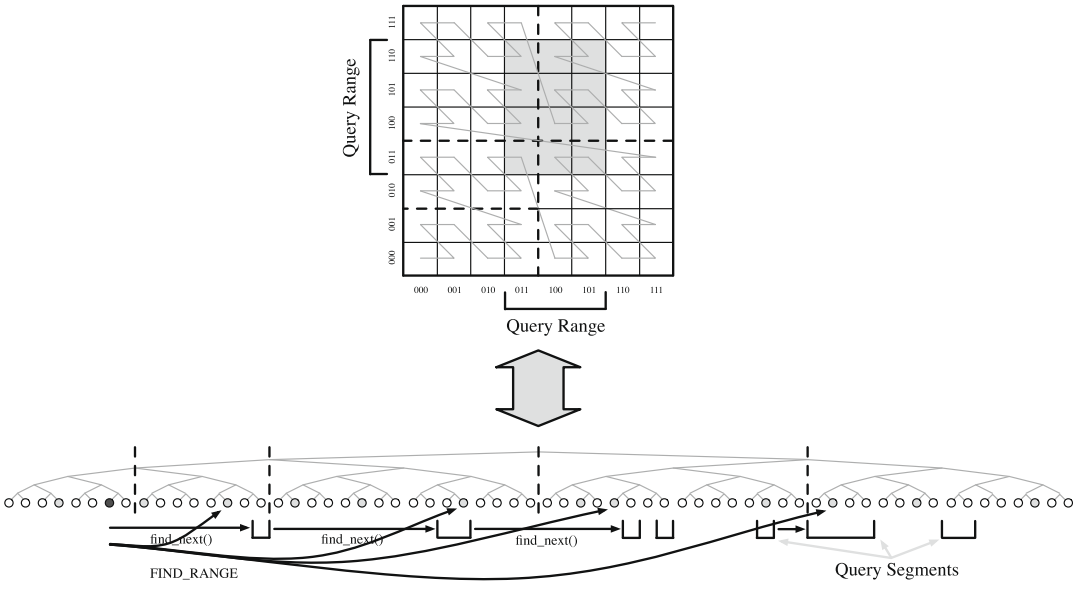

Fig. 4. A representation of a two-dimensional range query in both its *Z*-curve and DHT equivalents. Neighborhood borders are shown with dashed lines in both structures. A range query uses the `find_next()` function to move across neighborhoods, instead of going over each query segment in the DHT.

easily infer the locally stored values that should be replied. Local replicas are stored using their dimensional indices, therefore this requires a simple local database lookup. Moreover, the single-dimension scenario can now be considered as a sub-case of a unified algorithm, with `find_next()` implemented by incrementing Q, when Q is in-range. Also, note that Q_{start} is always returned by `find_next(range,-1)`.

4. Evaluation

Estimating the performance of range queries in a DHT requires considering numerous parameters that may affect their execution: network size, query size, data distribution, node distribution, number of dimensions used to index data and the overlay's replication factor. The latter, for example, controls the size of neighborhoods and as a result, their total number given a network size. We show that the node distribution does not play an important role in query resolution, as neighborhood forming is a recursive process that can tolerate large node concentrations. However, node and data distributions may affect the load balancing characteristics of the network, namely skew the amount of storage, processing and networking capacities demanded for some peer groups. Naturally, the query size controls how many neighborhoods will be visited by a query when the number of index dimensions is relatively low. However, when dimensionality escalates, the SFC may gradually lose its locality-preserving properties, thus map a relatively simple range to identifier segments that span a large percentage of the overlay, which in turn requires visiting even more peers.

As the range-query protocol additions do not alter the DHT's original routing tables, simple, point lookups in the network continue to cost about $\log N$ steps (for an overlay of N nodes). We assume that their range counterparts require a maximum of $\frac{S}{S_{max}} E \log N$ messages, where S is the length of identifier coverage, S_{max} the identifier space's size and E the total number of neighborhoods in the network. For multi-dimensional range queries, we consider the worst case where S corresponds to the total coverage of all query segments, including any gaps between them. Intuitively, the complexity of a range lookup is proportional to the number of neighborhoods it covers. As each neighborhood access is a standard lookup of $\log N$ cost, the term $\frac{S}{S_{max}}$ defines the percentage of branches that will be visited, depending on the size of the request. This percentage-based calculation implies a random distribution of nodes, however, it helps in gaining an insight on the overlay's performance characteristics, before discussing load-balancing issues. By all means, the worst case scenario is a range-query for *all* data keys, that has to go through all E neighborhoods, regardless of the node distribution.

In the following sections, we present both a formal and an empirical approach to determine E, before moving on to experimental results that indicate how the number of index dimensions and corresponding query sizes affect S. We then discuss on deployment and optimization tactics, as well as load-balancing techniques.

4.1. Counting the number of neighborhoods

To infer the total number of neighborhoods in a Kademlia network of size N and a replication factor of κ, we consider the following generalized problem.

Problem. Imagine a full binary tree of *unlimited* depth. The depth of the root is 0 and increases 1 for each level. Each node is a *bucket* that can hold a maximum of $\kappa - 1$ balls. When there are κ or more balls in a node, the bucket floor *breaks* and the bucket interior is connected to its successors. All balls slide randomly to either successor, as each one follows a predetermined but uniformly random (infinite) path.

Pour N balls in the root, $N \geqslant \kappa$. Buckets break and a tree is formed from all buckets that have been connected to the root. What is the average number of leaves of this tree?

Solution. Beginning with the root, the number of leaves to the connected tree is 1. For each breaking of a bucket, the number of leaves is increased by 1. Therefore, the average number of connected leaves is the average number of bucket breakings. The average number of breakings per bucket is the bucket's probability to break. Therefore, the average number of connected leaves equals the sum of to-break probabilities for *all* buckets in the unlimited tree.

At level d there are 2^d buckets, each having 2^{-d} probability to receive a ball reaching the level, as the overall distribution of balls through a level is uniform. Also, since distribution of balls across a level is uniform, each bucket on a level has the same breaking probability $P_{break}(N, \kappa, d)$.

The average of connected leaves is:

$$E(N,\kappa) = 2 + \sum_{d=1}^{\infty} 2^d P_{break}(N,\kappa,d) \tag{1}$$

Since $N \geqslant \kappa$, the root always breaks and contributes 1 leaf, counting itself being initially connected with itself.

Now, let P_x be the probability that exactly x of the N balls are predetermined to pass through a bucket in level d. The event can be constructed by selecting x out of N to put into the bucket and allow the other $N - x$ to freely go to any of the remaining $2^d - 1$ buckets:

$$P_x = \binom{N}{x} (2^d - 1)^{(N-x)} (2^{-d})^N$$

Then,

$$P_{break} = \sum_{x=\kappa}^{N} P_x = 1 - \sum_{x=0}^{\kappa-1} P_x$$

Finally, (1) is completed:

$$E(N,\kappa) = 2 + \sum_{d=1}^{\infty} 2^{-d(N-1)} \sum_{x=\kappa}^{N} \binom{N}{x} (2^d - 1)^{(N-x)}$$

Or, computationally more convenient when $\kappa \ll N$,

$$E(N,\kappa) = 2 + \sum_{d=1}^{\infty} 2^d \left(1 - 2^{-dN} \sum_{x=0}^{\kappa-1} \binom{N}{x} (2^d - 1)^{(N-x)}\right) \tag{2}$$

Assuming a perfect node placement, where all neighborhoods would have as close to $\kappa - 1$ nodes as possible, E would approximate $\frac{N}{\kappa-1}$. However, nodes are randomly placed in the identifier space, meaning that the *density* of neighborhoods will be less than the maximum. Knowing $E(N, \kappa)$, we construct the function that relates the average number of neighborhoods to $E_{min} = \frac{N}{\kappa-1}$. In Fig. 5 we demonstrate the *density multiplier* $D = \frac{E(N,\kappa)}{E_{min}}$ as a function of $\frac{\kappa}{N}$. It is evident that D oscillates with increasing amplitude as κ reaches N. Nevertheless, the region of the graph that is of particular interest in a Kademlia deployment is where $\kappa \ll N$. Our conjecture is that the axis of the oscillation is the line $D_{axis} = \frac{1}{\ln 2}$ and thus we generally use $E = 1.44 \frac{N}{\kappa-1}$ when calculating the number of neighborhoods in a typical overlay.

Moreover, empirical results from multiple simulations of neighborhood forming, suggest that this practical approximation of E holds even if the distribution of nodes is not completely uniform. In Fig. 6, we show the neighborhood count as a function of κ, for both a random and a skewed distribution of nodes in a two-dimensional identifier space of 20 bits per dimension. The two-dimen-

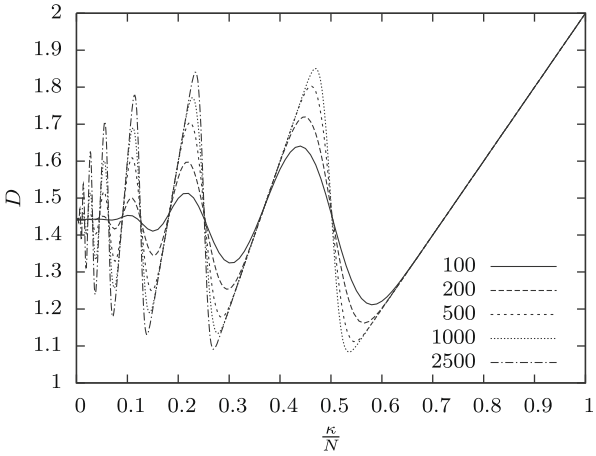

Fig. 5. Neighborhood density multiplier as a function of $\frac{\kappa}{N}$ for different values of N. When $\kappa \ll N$ the number of neighborhoods is about 44% larger than the optimal $\frac{N}{\kappa-1}$.

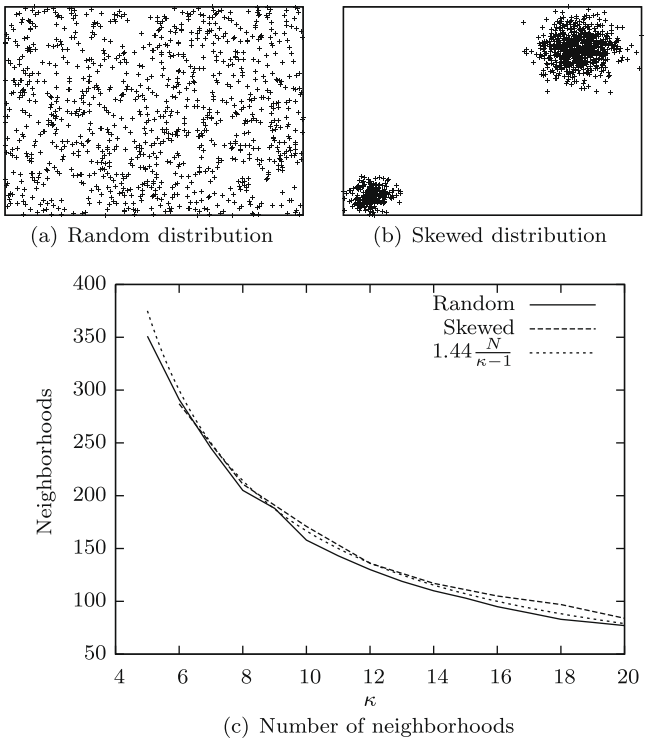

Fig. 6. Two example node distributions and the corresponding number of neighborhoods as a function of κ. Each layout shows 1000 nodes in a two-dimensional space of 20 bits per dimension.

sional space has been chosen for a better visualization of the resulting node densities in corresponding areas of the network. Node identifiers were transformed using the *Z*-order before being placed in Kademlia's binary structure. An interesting observation is that while respective plots of D for various non-random distributions exhibit different behavior from Fig. 5 and have a tendency to produce larger values, it still seems that:

$$\lim_{\frac{\kappa}{N}\to 0} D = D_{axis} = \frac{1}{\ln 2}$$

This conjecture is also supported by the fact that when we generalize Eq. (2) for *b-ary trees*:

$$E_b(N,\kappa,b) = b + (b-1)\sum_{d=1}^{\infty} b^d P_{nobreak}(N,\kappa,d,b)$$

$$P_{nobreak}(N,\kappa,d,b) = 1 - b^{-dN}\sum_{x=0}^{\kappa-1}\binom{N}{x}(b^d-1)^{(N-x)}$$

Simulation results suggest that for $\kappa \ll N, D_{axis}$ follows the pattern:

$$\lim_{\frac{\kappa}{N}\to 0} \frac{E_b(N,\kappa,b)}{E_{min}} = \frac{b-1}{\ln b}$$

4.2. Determining the effect of data dimensions

When a query refers to single-dimensional data, the percentage of visited neighborhoods is equal to the requested range of identifiers divided by the total size of the identifier space, S_{max}. However, when respective operations are applied to multi-dimensional values, the amount of visited neighborhoods is also affected by the number of index dimensions used. The correspondence of multi-dimensional ranges to continuous identifier segments seems to deteriorate when the dimensionality escalates and as a result, queries require visiting a larger number of smaller identifier groups, spread out wider across the network. To determine the amount of this effect, we simulated multiple range requests of varying lengths in a 10,000 peer overlay, counting each time the number of neighborhoods involved in collecting replies. Our simulation engine uses the *Z*-curve, so all results should be interpreted accordingly. While the Hilbert curve may perform comparatively better with the same index count, the analysis is representative of what effects should be expected at the corresponding dimensional limits of other SFCs.

In Fig. 7, we present the relationship between two percentages – the average visited neighborhoods for different amounts of queried ID space. Each run, for each dimension count, includes the result of about 10,000 queries, randomly generated to provide multiple samples for each point of the horizontal axis. All query boxes, or *hypercubes* in dimensions 4 and 5, have equal-length edges. Also κ is set to 20, so the total number of neighborhoods in all cases is about 750. It is evident that increasing the number of dimensions causes the querying node to visit all available neighborhoods faster. Although the two-dimensional case behaves very well, for four dimensions all neighborhoods are involved when the percentage of the queried ID space exceeds 70%, while for five dimensions this occurs much sooner, at about 23%.

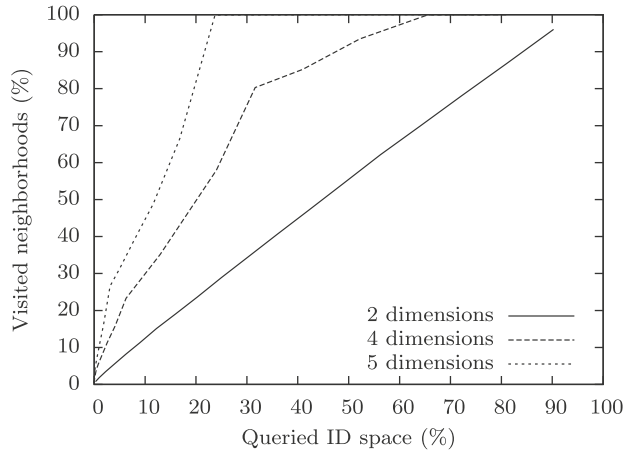

Fig. 7. Neighborhoods visited as a function of query size, for datasets of different dimensions.

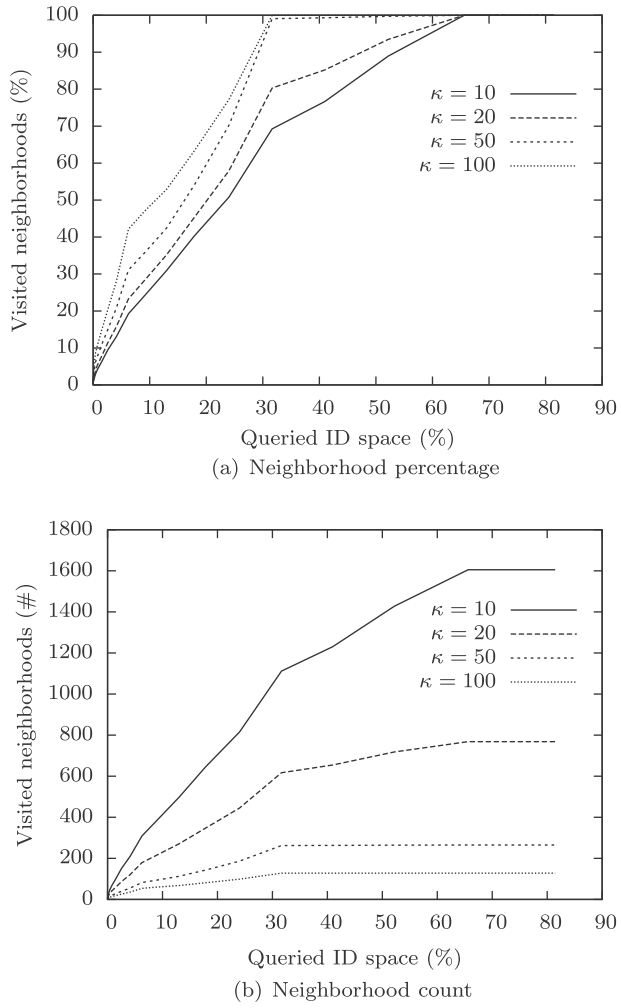

Fig. 8. Varying the neighborhood size while keeping the dimension count fixed to 4, in a network of 10,000 nodes.

The plots are also affected by the network's replication factor. Raising κ decreases the total number of neighborhoods – therefore, the number of hops through the overlay, but also makes subtrees larger, thus the querying process becomes coarse-grained and the percentage of network coverage bigger. When the neighborhoods are small, they may be more, but there is a higher probability that a query will skip some of them while traversing the query range from Q_{start} to Q_{end}. This is depicted in Fig. 8. Note that when $\kappa = 10$ the percentage of neighborhoods affected scales smoother, although the actual number of messaging hops is nearly double than when $\kappa = 20$. Even so, a query that floods the network, by requesting almost all available data, will visit only a small percentage of the total node population. The value of κ directly affects this percentage. Larger values may decrease the hop count and lower query times, but have side-effects discussed in the following section.

4.3. Discussion

Assuming a random distribution of nodes, except from the query coverage, the number of lookups required for each range query depends on N, κ and the number of dimensions used to reference values. The latter, as well as κ, have immutable values selected once when deploying the DHT. Choosing appropriate values may prove beneficial, especially if the general network properties are known or can be estimated beforehand.

In most deployments, κ is restrained from 10 to 20, in order to avoid over-replicating data. Given a network of 1000 nodes, $\kappa = 20$ and a request for all the identifier range, this may require nearly 75 lookups, which is manageable, both in terms of result latency and bandwidth consumption. As shown, when the node count increases by an order of magnitude, so does the number of peer groups. However, both from our experience and the study of related work, we can safely assume that range queries seldom refer to such large spaces. Usually, applications operate on a small percentage of the dataset, no more than 10%. Moreover, the execution of each range query can be easily optimized, either by further subdivision and parallelization at the DHT level, or by progressing sideways on corresponding neighborhoods.

Concerning the number of value indices, using a low dimension count maintains a fair analogy of query to network coverage. In general, due to limits of commonly used SFCs, managing more than 4–5 dimensions results in widely spread identifier ranges, even for queries that request only a small percentage of the multi-dimensional space. Therefore, readily available datasets may require some processing in order to scale the index space down. To this end, related literature suggests two methods that, depending on the index characteristics, can even be combined. First, some indices may be transformed to value attributes. When using this method, only a portion of the original indices is used in the DHT. Other attributes with a smaller range of values – i.e. binary properties, are stored along with the data. Queries are performed on the indices chosen and results are filtered based on their attributes, either at the DHT, or the application level. The second method suggests combining some indices into a single numerical space, but can only be used when queries are not expected or allowed to refer to corresponding ranges in respect to each other.

4.4. Balancing the load

Using an SFC for data placement, will result in the clustering of generated identifiers in correspondence to the semantical relationships of information in the multi-dimensional index space. So, unless the inserted dataset is randomly distributed in all dimensions, there will be irregular densities of value concentrations throughout the overlay. Although this does not affect the overlay's functionality in any way – as shown, both point and range lookup routing depends only on network size and κ, it may result in heavy loads from the nodes' perspective. If the data is skewed, some neighborhoods will have to handle more load than others. Load in this context can have two forms, namely increased requirements in storage availability and query processing capacity. Moreover, the latter is determined by the peer's computational power and network connectivity.

Load issues can arise if the dataset is large, if the expected query distribution is accordingly skewed, or even if the devices used to form the DHT are under-powered or connected via limited-bandwidth network links. The solutions proposed so far for handling the problem are either to enforce nodes join in a predetermined network hotspot or provide a means for moving existing peers to congested areas. Both tactics try to match the data irregularity with the node distribution, thus keeping a fair proportion of peers to values in any DHT neighborhood, and require an internal or external mechanism to observe the key-value pair distribution and define thresholds on the densities of items. If the full data collection, or its locality characteristics, are well-known beforehand, it should be straight-forward to derive matching node identifiers. The neighborhood count will not be affected (Fig. 6), but as neighborhood sizes will not be uniform, the average number of lookups per query cannot be easily pre-estimated. It will definitely be larger than $\frac{S}{S_{max}}E$, although bound by E.

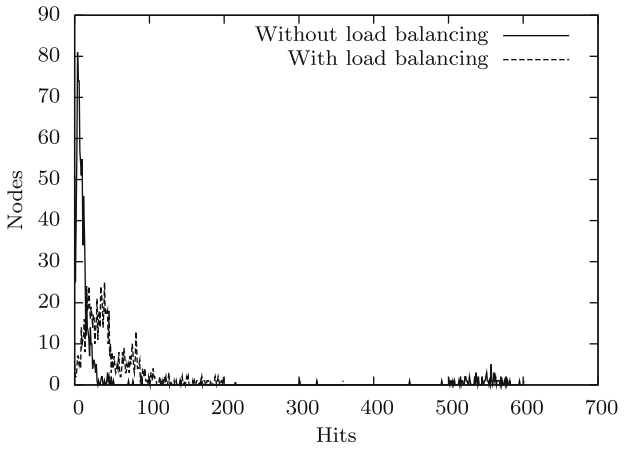

Fig. 9. Number of nodes per number of hits, while performing 10,000 range queries in a network of 1000 peers. Visiting a random node in each neighborhood results in a better balancing of the imposed load.

Also, moving a peer to a new position in the overlay is best accomplished via *virtual peers* – by assigning multiple identifiers to the same node, therefore allowing a peer to be present at numerous neighborhoods. Otherwise, literally disconnecting and then reconnecting the same node at a different part of the DHT may require additional maintenance operations (routing table updates and invalidations), which in turn introduce latencies depending on the nature of the protocol's fault detection and stabilization functions. Our implementation platform already allows for deploying multiple nodes per physical machine. In controlled deployments, this feature paired with an external observer that monitors the data distribution can help in deriving an automated load-balancing mechanism, that reassigns peers from less congested areas to overloaded segments of the identifier space as virtual instances.

We argue, however, that in highly dynamic setups load estimation cannot solely rely on objective, unbiased metrics. All the aforementioned techniques try to assign query operations to different nodes, but do not account their particular characteristics, although the overall population may be heterogeneous. Load measurement and distributed propagation of respective information in the context of a DHT is a challenging problem on its own, whose analysis and applications extend beyond the scope of this paper.

In the proposed overlay, an analogous effect can be produced by exploiting the fact that contacting *any* node in a neighborhood will yield the same results. In Fig. 9, we show the number of nodes per number of visits, in two runs of the same scenario that involved 10,000 range queries for 5% of the index space in an overlay of 1000 participants. When processing each query directionally, the load is concentrated to the nodes closest to neighborhood borders. These are the peers that are visited 500–600 times. When randomizing the node selection, the two separate areas of the initial graph are merged into one – the same number of hits are better distributed among available peers. Corresponding results are also obtained when scaling the network and query sizes or the number of range operations.

5. Motivation

5.1. Case study: the Gredia middleware for data management

To present a concrete example of how our work could facilitate efficient query processing, let us consider the *Gredia* project, part of the European Commission's IST 6th Framework Programme (FP6-34363) [5]. The Gredia platform provides a Grid-based infrastructure that allows developers, professionals and simple users alike to share annotated multimedia content. Although a research project, many of the scenarios implemented reflect on realistic use cases driven by the project's business partners (Deutche Welle, Popso, etc.). A representative expected usage pattern is as follows: Various news agencies have created a joint data repository on the Grid, where employees (journalists, photographers, editors) or citizens can store, search and download news items, be it photos, videos or text according to their access rights. Assume that just minutes after the riots in Athens break out, a Greek journalist on scene captures a video of the protests and uploads it on the Grid. At the same time, many by-standees capture photos and clips of the events. Hundreds of journalists around the world need to be able to quickly locate and download this content in order to include it in the news-break of their broadcast stations. The Gredia platform provides an efficient means of storing and searching for data, as well as of orchestrating any kind of editorial, reviewing, or publishing process that will result in the proper announcement of a news item.

Fig. 10 shows the Gredia middleware architecture, as described in [36]. It consists of various components, among which three overlays: The *metadata overlay* stores the metadata associated with each multimedia file, while the other two peer-to-peer structures provide distributed replica management and collective file transfer services for the actual data. In practice, a participating node can be part of multiple overlays, depending on its capabilities and usage intentions. Nevertheless, each is studied independently and as a unit constitutes a platform for researching and developing appropriate algorithms and protocols. Other components carry out the authentication and authorization processes, by mapping users to different Virtual Organizations (VOs) and filtering requests according to their corresponding credentials. For example, a data file can be stored in a storage node, only if the user has the appropriate permissions.

The work presented here focuses on the metadata facility. Multimedia content is annotated along multiple dimensions as it deemed important to characterize the file and make it searchable upon. Both editors and journalists are allowed to fully customize their searches over a potentially vast repository: Users should be able to perform efficient point and range queries over the multiple attributes that index each file. In all usage scenarios, it is of great importance, from both a business and research perspective, that queries on multiple dimensions with any combination of values (*all*, point or range) be processed quickly. With our proposed scheme, the number of visited peers is dynamically minimized and consequently so is the bandwidth utilized to collect answers. Moreover, fault-tolerance is automatically handled by the underlying DHT's inherent replication mechanism, without influencing the effectiveness of the query engine.

5.2. Implementing the metadata overlay

In the data management layer, we consider two basic entities for each unit of information – the "*data file*" containing the actual content and the "*metadata file*" with corresponding annotations according to a pre-defined set of attributes to be indexed. Metadata files are created, distributed and stored in human-readable XML format. A *metadata module* of the middleware, implements the necessary interface with the overlay and handles all metadata insertion, search and retrieval operations from the application's perspective. The module is responsible for encoding the values of attributes into bit representations that will be interpreted by the overlay as dimensional indices. Storing a metadata file, requires parsing the XML elements and extracting attribute values, before producing such indices and placing the file into the network.

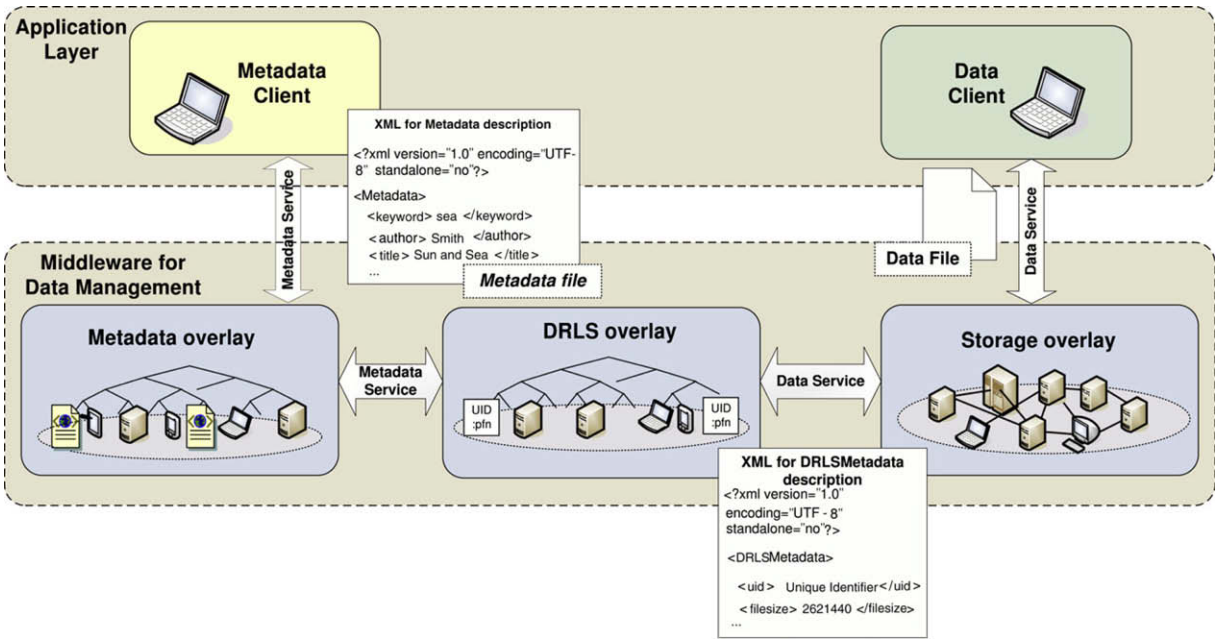

Fig. 10. Overview of the Gredia middleware layer.

Invoking the search operation and collecting results for a query, except from encoding contained attributes, may also require appropriate transformations in order to match the ordering of indices in the overlay.

Applications implemented in the context of the Gredia project assume three basic types of attributes, namely *string*, *date* and *categorical*. The pre-defined set of attributes to be indexed is considered to form a multi-dimensional space, where each attribute corresponds to a respective dimension. A problem to tackle when deploying the proposed multi-dimensional indexing scheme has been the segmentation of each dimension according to the number of possible values for each attribute. Since the SFC-based method for enabling range queries over multiple dimensions presumes constant-bit indices, we have developed corresponding mapping functions for each attribute type.

The categorical attributes are easily manageable, since their range of possible values is known in advance and can be directly mapped to the available index space. Although range queries are not a common case for categorical types, we take advantage of the imposed ordering at the dimensional level to enable the retrieval of metadata files characterized by more than one category. In this case, the query is rephrased by the metadata module according to the ordering convention followed when encoding. It is possible that multiple distinct values are joined in one or more ranges when the search query reaches the overlay.

For string and date types, we use a simple character-coding scheme to produce bit representations. However, the number of encoded characters that comprise the dimensional index depends on the number of bits available for the attribute at the overlay level. Taking into account that the number of index bits in each dimension is equal, the cardinality of the SFC is dominated by the most numerous attribute. However, instead of scaling all attribute representations up, we select an approximation order of the used curve that requires a reasonable amount of bits – depending on the application – and then "crop" encoded values accordingly. Our goal is to avoid using a large number of bits, so searches refer to more restricted search spaces, thus end up to less nodes and are served more efficiently. Further filtering may follow in case the string value contains more than the allowed characters. The encoding results in alphabetical ordering of strings on the corresponding dimensions. For date types we first order year, then month, day, hour and minute.

In the Gredia project, two basic applications are implemented, namely the media and the banking applications. The former requires a limited number of attributes to be indexed and thus we use a two-dimensional query space, composed of a string for customer name and a numerical customerID. On the other hand, the media application requires multiple dimensions, since a multimedia file can be described by various attributes that may be correlated to the content of the file, its type, coding format, etc. To avoid increasing the number of mapped attributes excessively, we decided on the most critical ones, which are expected to be commonly employed by media-related users. The selection is shown in Table 1. Naturally, all file attributes are still included in the metadata file for application-level selection (i.e. attributes defined by the MPEG-7 standard [37]). It has been observed, that categorical types with limited values can safely be omitted, so as to reduce the number of dimensions and increase the efficiency of range queries. Filtering can also occur directly at the overlay nodes, by sending the original query along with the transformed range representation. This technique, which effectively results in using the most representative attributes for routing and all for local result filtering, requires incorporating parts of the metadata module at the peer level, but helps in easily achieving good dimensional scalability while distributing the workload to network participants.

In the media example, a point query requires a specific value for all of the attributes presented in Table 1 and should be answered by a single metadata file. If the file exists, it is retrieved by a simple Kademlia lookup. Range queries, such as '*Get the metadata files of*

Table 1
Indexed attributes in the media application.

Attribute	Type	Example
Author	String	Smith John
Date created	Date	2008.02.01
Date modified	Date	2009.05.01
Type	Categorical	Audio, text, video
Category	Categorical	Entertainment, business, politics, weather, travel, health

the videos filmed by Smith John, which were created from 2008.09.28 until 2008.10.25 and modified from 2008.09.29 until 2008.10.30 regarding weather' are routed according to the described protocol. Moreover, the user may perform a '*'-lookup for one or more attributes. A '*' is equivalent to a range with all possible values in the corresponding dimension and is translated to this by the metadata module.

We have implemented a working prototype of the discussed range-query protocols as an extension to Kademlia in our own peer-to-peer evaluation and deployment platform, which we call *PeerPlatform*. PeerPlatform is structured as a Python module that can run either stand-alone or as a library embedded into other applications. Each platform instance can host multiple nodes, from the same or different protocol family. The rest of the Gredia middleware, mostly written in Java, interfaces with the underlying peer-to-peer overlays via an open network protocol that is directly understandable by the PeerPlatform's notification-driven infrastructure. We also provide a simple web-service interface to send point or range *get* and *put* commands to a platform's locally running peers.

6. Conclusion and future work

Supporting range queries in structured peer-to-peer systems requires both an appropriate data-placement scheme, as well as an intelligent query engine that imposes the least possible computational and management overhead at participating peers. Our proposed protocol succeeds in applying multi-dimensional range-query techniques in a DHT context, while taking into account value replication that is provided internally by many modern overlays. Using a Kademlia network as an example, we show how to group nodes into replica neighborhoods, by exploiting information already available in the peers' routing tables. Range queries are then resolved by following a fault-tolerant path through neighborhoods, automatically bypassing duplicate values. The protocol, being dependent on routing table information, dynamically adjusts query execution as respective information changes due to node behavior or network conditions.

Detailed analysis of the effects introduced by the various overlay parameters, lead to a thorough understanding of the system's functionality and allow us to predict its performance and scalability characteristics, thus preemptively tune deployments according to expected datasets and usage patterns. Most importantly, we have theoretically calculated and verified through simulations the number of neighborhoods E that are produced by Kademlia peers, as a function of network size and κ, the overlay's replication factor. We also present how the number of dimensions used to index data, influences the slope of the curve that binds query size with the number of hops required to complete the request – up to a maximum of E. Although in our scheme the data distribution does not have a direct impact on query execution, skewed datasets may require incorporating methods for load-balancing among available nodes. We elaborate on how to exploit peer neighborhoods in this direction and present corresponding simulation results.

Finally, we show how our implementation prototype fits in with the data management layer of the Gredia project, describe application examples and discuss on upper-layer range query management and processing.

References

[1] H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Looking up data in P2P systems, Communications of the ACM 46 (2) (2003) 43–48.

[2] P. Maymounkov, D. Mazières, Kademlia: a peer-to-peer information system based on the XOR metric, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), Cambridge, USA, March 2002, pp. 53–65.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the 2001 ACM SIGCOMM Conference, San Diego, USA, August 2001, pp. 149–160.

[4] R. Cox, A. Muthitacharoen, R.T. Morris, Serving DNS using a peer-to-peer lookup service, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), Cambridge, USA, March 2002, pp. 155–165.

[5] Grid Enabled access to rich media content (GREDIA) IST project [Online], <http://www.gredia.eu>.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, A scalable content-addressable network, in: Proceedings of the 2001 ACM SIGCOMM Conference, San Diego, USA, August 2001, pp. 161–172.

[7] A.I.T. Rowstron, P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Springer-Verlag, London, UK, 2001, pp. 329–350.

[8] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 41–53.

[9] A. Gupta, D. Agrawal, A.E. Abbadi, Approximate range selection queries in peer-to-peer systems, in: Proceedings of the First Biennial Conference on Innovative Data Systems Research, 2003.

[10] A. Andrzejak, Z. Xu, Scalable, efficient range queries for grid information services, in: Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden, September 2002, pp. 33–40.

[11] J. Gao, P. Steenkiste, An adaptive protocol for efficient support of range queries in dht-based systems, in: ICNP '04: Proceedings of the 12th IEEE International Conference on Network Protocols, IEEE Computer Society, Washington, DC, USA, 2004, pp. 239–250.

[12] C. Zheng, G. Shen, S. Li, S. Shenker, Distributed segment tree: support of range query and cover query over dht, February 2006.

[13] A. Crainiceanu, P. Linga, J. Gehrke, J. Shanmugasundaram, Querying peer-to-peer networks using p-trees, in: WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, ACM, New York, NY, USA, 2004, pp. 25–30.

[14] S. Ramabhadran, S. Ratnasamy, J.M. Hellerstein, S. Shenker, Brief announcement: Prefix Hash Tree, in: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC '04), St. John's, Newfoundland, Canada, New York, NY, USA, 2004, p. 368.

[15] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, J. Hellerstein, A case study in building layered dht applications, SIGCOMM Computer Communication Review 35 (4) (2005) 97–108.

[16] E. Caron, F. Desprez, C. Tedeschi, Enhancing computational grids with peer-to-peer technology for large scale service discovery, Journal of Grid Computing 5 (3) (2007) 337–360.

[17] H. Zhuge, X. Sun, J. Liu, E. Yao, X. Chen, A scalable P2P platform for the knowledge grid, IEEE Transactions on Knowledge and Data Engineering 17 (12) (2005) 1721–1736.

[18] A. Datta, M. Hauswirth, R. John, R. Schmidt, K. Aberer, Range queries in trie-structured overlays, in: Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005), Konstanz, Germany, August 2005, pp. 57–66.

[19] J. Aspnes, G. Shah, Skip graphs, ACM Transactions on Algorithms 3 (4) (2007) 37.

[20] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, Communications of the ACM 33 (6) (1990) 668–676.

[21] A. González-Beltrán, P. Milligan, P. Sage, Range queries over skip tree graphs, Computer Communications 31 (2) (2008) 358–374.

[22] A.R. Bharambe, M. Agrawal, S. Seshan, Mercury: supporting scalable multi-attribute range queries, SIGCOMM Computer Communication Review 34 (4) (2004) 353–366.

[23] T. Schütt, F. Schintke, A. Reinefeld, Range queries on structured overlay networks, Computer Communications 31 (2) (2008) 280–291.

[24] R. Bayer, The universal B-tree for multidimensional indexing: general concepts, Lecture Notes in Computer Science 1274 (1997) 198–209.

[25] H. Tropf, H. Herzog, Multidimensional range search in dynamically balanced trees, Applied Informatics 2 (1981) 71–77.

[26] J. Lawder, P. King, Using space-filling curves for multi-dimensional indexing, Lecture Notes in Computer Science 1832 (2000) 20–35.

[27] B. Moon, H. Jagadish, C. Faloutsos, J. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve, IEEE Transactions on Knowledge and Data Engineering 13 (1) (2001) 124–141.

[28] C. Schmidt, M. Parashar, Squid: enabling search in dht-based systems, Journal of Parallel and Distributed Computing 68 (7) (2008) 962–975.

[29] J. Lee, H. Lee, S. Kang, S.M. Kim, J. Song, CISS: an efficient object clustering framework for DHT-based peer-to-peer applications, Computer Networks 51 (4) (2007) 1072–1094.

[30] P. Ganesan, B. Yang, H. Garcia-Molina, One torus to rule them all: multi-dimensional queries in P2P systems, in: Proceedings of the 7th International Workshop on the Web and Databases (WebDB '04), Paris, France, 2004, pp. 19–24.

[31] E. Tanin, A. Harwood, H. Samet, Using a distributed quadtree index in peer-to-peer networks, The VLDB Journal 16 (2) (2006) 165–178.

[32] Y. Shu, B.C. Ooi, K.-L. Tan, A. Zhou, Supporting multi-dimensional range queries in peer-to-peer systems, in: Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005), Konstanz, Germany, August 2005, pp. 173–180.

[33] D. Li, J. Cao, X. Lu, K.C. Chen, Efficient range query processing in peer-to-peer systems, IEEE Transactions on Knowledge and Data Engineering 21 (1) (2009) 78–91.
[34] D. Li, X. Lu, J. Wu, FissionE: a scalable constant degree and low congestion DHT scheme based on Kautz graphs, in: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005), vol. 3, March 2005, pp. 1677–1688.
[35] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, R. Bayer, Integrating the UB-Tree into a Database System Kernel, in: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt, September 2000, pp. 263–272.
[36] A. Asiki, K. Doka, I. Konstantinou, A. Zissimos, D. Tsoumakos, N. Koziris, P. Tsanakas, A grid middleware for data management exploiting peer-to-peer techniques, Future Generation Computer Systems 25 (4) (2009) 426–435.
[37] MPEG-7 Overview, International Organization For Standardisation, 2004 [Online], <http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm>.