Hamming DHT: An Indexing System for Similarity Search

Rodolfo da Silva Villaça[1], Luciano Bernardes de Paula[2], Rafael Pasquini[3], Maurício Ferreira Magalhães[1]

[1]Department of Computer Engineering and Industrial Automation (DCA)
School of Electrical and Computer Engineering (FEEC)
State University of Campinas (UNICAMP)
Campinas – SP – Brazil

[2]Federal Institute of São Paulo (IFSP)
Bragança Paulista – SP – Brazil

[3]Faculty of Computing (FACOM)
Federal University of Uberlândia (UFU)
Uberlândia – MG – Brazil

{villaca,mauricio}@dca.fee.unicamp.br, lbernardes@ifsp.edu.br,

pasquini@facom.ufu.br

Abstract. *The semantic meaning of a content is frequently represented as content vectors, where each dimension represents an attribute of this content. For instance, these attributes may represent keywords in a text, colors in a picture, profile information in a social network, etc. One main challenge in this semantic context is the storage and retrieval of similar contents. This way, a new Distributed Hash Table (DHT), called Hamming DHT, is proposed in this paper. The Hamming DHT leverages the Locality Sensitive Hashing (LSH) functions, specially the Random Hyperplane Hashing (RHH) family, in order to improve the performance of searches for similar content (similarity search). The proposed DHT propitiates a scenario in which similar contents are stored in peers nearly located in the DHT indexing space. In a comparison with Chord, the experimental results indicate the effectiveness of our proposal in terms of the number of hops required to retrieve similar content, and shows that the proposed DHT is able to aggregate them, improving the similarity search.*

1. Introduction

The advent of Peer-to-Peer (P2P) networking changed the way information is distributed, providing to users the experience of sharing contents of their interest. In theory, P2P networks constitute a cooperative system built by the users, in which an unlimited amount of information from different areas can be accumulated, creating a virtuous cycle for attracting new users. However, the search for information of interest was not a trivial problem, compromising the P2P virtuous cycle. Firstly, given a search, what are the contents really related to it? Afterwards, given a search, where are such contents stored? Essentially, in the first P2P systems, users were forced to perform several searches, providing accurate keywords in order to find the desirable contents. At the same time, such mechanisms were based on the flooding of search messages, causing an elevated exchange of control messages.

As part of the evolution, the second generation of P2P systems introduced the use of Distributed Hash Tables (DHTs), aiming to facilitate the search and retrieval of content. Through a structured overlay network, a large volume of data can be organized using unique content identifiers, that are generated by using base hash functions, such as MD5 and SHA-1. In this solution, peers participating in the DHT also possess an identifier and are responsible for storing a portion of the overall identity space. As presented in Chord [Stoica et al. 2003], the portion assigned to each peer is related to its position in a virtual ring that organizes the identity space. Afterwards, by using key-value primitives, like *put(k,v)* and *get(k)*, information can be stored and retrieved, avoiding the use of flooding mechanisms. However, the use of MD5 and SHA-1 leads to an homogeneous distribution of the identifiers in the identity space of the DHT, does not preserving the semantic of similar contents in terms of the location where they are stored. For example, two very similar contents can be stored in two far away peers, making difficult the search and retrieval of them. In short, the search for contents of interest is still not efficient given the scattering of information in the DHT and, also, given the need of providing the exact content identifier for the *get(k)* primitive to retrieve the content.

An alternative to improve the searching for similar contents is to organize them based on their similarities. For this purpose Locality Sensitive Hashing (LSH) functions can be applied, instead of using MD5 or SHA-1. The benefit of organizing content identifiers by similarity is to make it easier to search and retrieve them in the DHT. It is important to mention that this paper considers a search for similar contents as being a search in which a set of contents, similar to a given query, is returned. Basically, the searching objective is to retrieve a set of similar contents greater than or equal to a similarity index. Consequently, it is fundamental to assure that similar identifiers represent similar contents in order to boost the search/retrieval experience of users.

In the literature, a common way of measuring the similarity between contents is to use the cosine of the angle between content vectors, in which each dimension represents a unique characteristic of the content, such as keywords in a text [Berry et al. 1999], color histogram in a picture [Kong 2009] or fields in a multidimensional data structure [March and Teo 2005]. Generally, in a content vector schema, the similarity between contents can be measured in two ways: using the Euclidean distance or the cosine of the angle between the content vectors [Qian et al. 2004]. Space Filling Curves (SFC), such as the Hilbert curve, are often used to cluster similar contents in the Euclidean distance space, but as explained in [Indyk and Motwani 1998], this is an expensive choice due to the curse of dimensionality.

On the other hand, the Locality Sensitive Hashing (LSH) functions are capable of creating hash keys, maintaining the similarities between their input data, i.e., similar items are mapped to similar hash identifiers, with high probability, reducing the curse of dimensionality [Indyk and Motwani 1998]. Among the existent LSH functions, the Random Hyperplane Hashing (RHH) [Charikar 2002] is a family of LSH functions whose similarity corresponds to the cosine of the angle between vectors. As shown in a previous work [de Paula et al. 2011], using RHH, the similarity between contents can be represented by the Hamming distance of their identifiers with an elevated accuracy level. In this context, the next step is to provide an indexing system capable of extracting benefits from this hashing mechanism, as it can not be fully explored in traditional DHTs, like

Chord, because peers and content are organized in a crescent order of identifiers.

Hence, this paper proposes a DHT whose structure is able of reflecting the similarity provided by the Hamming distance between content identifiers, generated by using the RHH function. Essentially, the proposed DHT establishes fingers between peers according to the Hamming distance of their identifiers and, in this way, creates clusters of similar peers to assure a small number of hops between them. Another important aspect related to the proposed DHT is that, differently of the traditional DHTs that organize the virtual ring based on the natural (crescent order) binary code, the identity space of the proposed DHT is organized according to the Gray code sequence, where two successive identifiers differ in only one bit. Thus, the slices of content identifiers are defined according to the Gray code sequence, assigning fractions of the content to the peers existent in the DHT. To demonstrate its effectiveness, this paper shows, using experimental results in a comparison with Chord, that the proposed Hamming DHT is capable of aggregating similar contents in peers near located in the DHT structure. In short, the recall of information is improved, since the number of hops required to retrieve similar contents decreases.

The remainder of this paper is organized as follows: Sections 2, 3 and 4 compose the related work literature, where Section 2 justifies the use of cosine similarity by comparing it with the Euclidean distance similarity; Section 3 briefly introduces the RHH hash functions and the usage of the Hamming distance; and Section 4 presents the clustering properties of the Gray code. In the sequence, Section 5 details the proposed DHT, describing the process for establishing fingers according to the Hamming distance and how it operates under the Gray code sequence. Section 6 presents the evaluation results, comparing the proposed DHT with Chord. Section 7 concludes this paper.

2. Cosine and Euclidean distance similarity

This section justifies the option for adopting the cosine similarity to generate the content identifiers, instead of using the Euclidean distance. To this aim, consider an application scenario where documents are represented by 2-dimensional content vectors, where each dimension is represented by the keywords ‘network’ and ‘protocol’. Basically, the number of occurrences of each keyword in a given text forms a tuple that represents its content vector. For example, consider the scenario depicted in Figure 1, where three documents are stored and identified by the tuples DOC1(1,6) - 1 occurrence of keyword ‘network’ and 6 occurrences of keyword ‘protocol’, DOC2(3,2) - 3 occurrences of keyword ‘network’ and 2 occurrences of keyword ‘protocol’ and DOC3(5,5) - 5 occurrences of keyword ‘network’ and 5 occurrences of keyword ‘protocol’.

Based on the three documents available in the indexing system of Figure 1, consider that a user issues a query using a text in which the term ‘network’ occurs two times and the term ‘protocol’ occurs three times, as represented by the tuple QUERY(2,3) in Figure 1. If the indexing system considers the Euclidean distance to measure the similarity existent between the QUERY and the three available documents, DOC2 will be the more similar document, since it is closer to QUERY using the Euclidean distance metric. In essence, DOC2 contains the same number of keywords present in the QUERY, even though the term ‘network’ occurs more frequently than the term ‘protocol’. In a simplistic interpretation, such behavior can indicate that DOC2 is a text more focused on the ‘network’ than in the ‘protocol’ aspect, contrary to our interest.

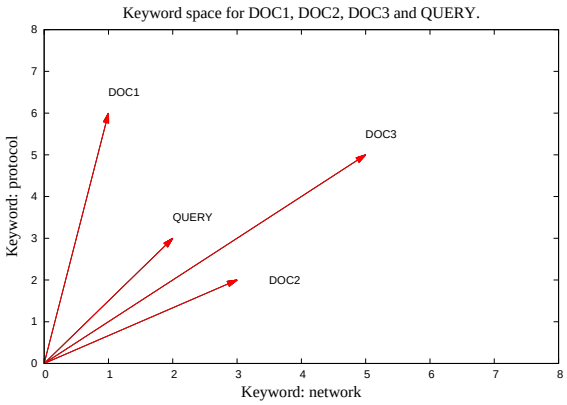


Figure 1. Keyword space for DOC1, DOC2, DOC3 and QUERY.

Conversely, if the indexing system considers the cosine of the angle existent between the vectors, DOC3 will be the more similar document. Note that in DOC3 both keywords are more frequent than in the QUERY, showing a slight difference of scale between them. However, such difference of scale can be interpreted, for example, as DOC3 being a longer text about the same subject of the QUERY, as exposed in [Berry et al. 1999], indicating a more accurate level of similarity and justifying the option of this work in adopting it to generate the content identifiers.

The next section presents the Random Hyperplane Hashing function and shows how the content identifiers generated are capable of reflecting similarity, measured by the Hamming distance.

3. The Random Hyperplane Hashing function and the Hamming Distance

The Locality Sensitive Hashing (LSH) functions allow to represent content vectors by reducing their dimensions without losing similarity. These functions are useful in the solution of the *k* nearest neighbors search problem, which is related to the search for the *k* nearest neighbors of a point, i.e., the query, in an indexing space [Indyk and Motwani 1998]. As mentioned before, the Random Hyperplane Hashing (RHH) is a family of LSH functions, whose similarity function corresponds to the cosine of the angle between vectors. In this context, Charikar [Charikar 2002] presents a hashing technique where the execution of a given LSH function returns a single bit, and the execution of a set of LSH functions is concatenated to generate the hash value of a vector. Such process is summarized in the sequence:

Given a set of size m of random vectors $\overrightarrow{r} \in \Re^d$, selected from a standard normal distribution, and a content vector $\overrightarrow{u} \in \Re^d$, a hash function $h_{\overrightarrow{r}}$ is defined as follows:

$$h_{\overrightarrow{r}}(\overrightarrow{u}) = \begin{cases} 1, \text{if } \overrightarrow{r} \cdot \overrightarrow{u} \geq 0 \\ 0, \text{if } \overrightarrow{r} \cdot \overrightarrow{u} < 0 \end{cases}$$

For each $h_{\overrightarrow{r}}(\overrightarrow{u})$ one bit is generated, and the results of m $h_{\overrightarrow{r}}(\overrightarrow{u})$ are concatenated to compose a *m*-bit hash key for the content vector $\overrightarrow{u}$. Then, for two content vectors $\overrightarrow{u}, \overrightarrow{v} \in \Re^d$, the probability of generating similar values is a function of the cosine of the angle between $\overrightarrow{u}$ and $\overrightarrow{v}$, as shown in the sequence:

$$\mathbf{Pr}\{h_{\overrightarrow{r}}(\overrightarrow{u}) = h_{\overrightarrow{r}}(\overrightarrow{v})\} = 1 - \frac{\theta(\overrightarrow{u}, \overrightarrow{v})}{\pi}$$

Consequently, as more similar two vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ are, the more likely the generated keys, resultant of $m\ h_{\overrightarrow{r}}(\overrightarrow{u})$ and $m\ h_{\overrightarrow{r}}(\overrightarrow{v})$, will share common bits, leading to two identifiers close in the Hamming distance. It is important to mention that the m $h_{\overrightarrow{r}}$ executions follow the same order in the set of random vectors $\overrightarrow{r}$, in order to generate similar content identifiers.

As an example, assuming that an application uses an identifier space of 8 bits, it is necessary to generate a set of $m = 8$ random vectors $\overrightarrow{r}$, and to concatenate the returned bits of the $m\ h_{\overrightarrow{r}}$, in order to extract the 8-bit content identifiers for each content vector. In the scenario previously presented in Figure 1, 8-bit content identifiers for the documents DOC1, DOC2, DOC3 and QUERY could be defined as presented in Table 1. As can be seen, DOC3, the more similar text to QUERY based on the cosine of the angle metric, has the content identifier with the smallest Hamming distance (D_h) to the QUERY identifier, i.e., DOC3 presents $D_h = 1$ to QUERY, corresponding to 87.5% of bits matching between both identifiers.

Table 1. Text identifiers, Hamming distance and Similarity level to the QUERY.

Text	Identifier	D_h	Similarity
QUERY	01010101	0	1
DOC3	11010101	1	0.875
DOC2	01111101	2	0.75
DOC1	11111111	4	0.5

4. Gray codes

This section briefly introduces the properties and definitions of the Gray codes, where two successive codewords differ in exactly one bit position, i.e., their Hamming distance is equal to 1. Thus, if these codewords were generated by the RHH family of LSH functions, successive codewords will represent similar contents.

A *n*-bit binary Gray code G_n can be represented as a $2^n \times n$ binary matrix, with each row being a codeword. If G_n is a *n*-bit Gray code, then G_{n+1} is a (*n*+1)-bit Gray code, which is produced by: 1) concatenating the G_n codewords to the G_n codeword in reverse order; 2) prefixing G_n codewords with the bit 0 and 3) prefixing the codewords in the reflected G_n code with the bit 1. Hence, recursively applying this method, the reflected binary Gray code is obtained which is trivially defined as G_1. To illustrate this method, Figure 2 shows the definitions for the binary reflected Gray codes G_1, G_2, G_3, G_n and G_{n+1}.

In order to create a virtual ring structure which follows the binary reflected Gray code sequence, it is possible to convert a conventional ring, organized using the sequential order of identifiers, to the Gray code sequence and vice versa. Such conversions can be performed with simple CPU operations such as XOR, shift right and comparisons to 0, as demonstrated in the Algorithms 1 and 2.

Essentially, the adoption of the Gray code in the proposed DHT is justified by its ability of successively positioning similar contents. In this scenario, performing searches

$$G_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad G_2 = \begin{bmatrix} 00 \\ 01 \\ 11 \\ 10 \end{bmatrix} \qquad G_3 = \begin{bmatrix} 000 \\ 001 \\ 011 \\ 010 \\ 110 \\ 111 \\ 101 \\ 100 \end{bmatrix} \qquad G_n = \begin{bmatrix} G_n[0] \\ G_n[1] \\ \dots \\ G_n[2^n - 1] \end{bmatrix} \qquad G_{n+1} = \begin{bmatrix} 0G_n[0] \\ 0G_n[1] \\ \dots \\ 0G_n[2^n - 1] \\ 1G_n[2^n - 1] \\ \dots \\ 1G_n[1] \\ 1G_n[0] \end{bmatrix}$$

Figure 2. Definitions for the binary reflected Gray codes G_1, G_2, G_3, G_n and G_{n+1}.

Algorithm 1: Binary to Gray code conversion.

Input : Binary codeword, $(binary)$.
Output: Gray codeword, $(gray)$.
$gray \leftarrow binary \oplus (binary \gg 1)$

Algorithm 2: Gray to Binary code conversion.

Input : Gray codeword, $(gray)$.
Output: Binary codeword, $(binary)$.
$binary \leftarrow gray$
$i \leftarrow (gray \gg 1)$
while $i \neq 0$ **do**
  $binary \leftarrow (binary \oplus i)$
  $i \leftarrow (i \gg 1)$
end

for contents that have identifiers with long bit strings in common (small Hamming distances), is facilitated given the better aggregation of the identifiers according to their Hamming distances. Faloutsos [Faloutsos 1988] analytically proves that this clustering property of the Gray codes is never worse than the clustering property of the binary codes and, in the best cases, it is up to 50% better.

5. The Hamming DHT

This section presents our proposal of a new DHT, called Hamming DHT, whose consistent hashing approach and the join and leave procedures are similar to Chord. As mentioned before, the main differences between the Hamming DHT and Chord are: 1) the use of the Gray codes in the organization of the identifiers in the ring and 2) the establishment of fingers based on the Hamming distance. Thus, the main goal of this proposal is to show that the use of Gray codes in conjunction with the Hamming distance facilitates the similarity search, when contents are indexed using the RHH function.

Regarding the related literature, Hycube [Olszak 2010] is an example of DHT that uses Hamming as a distance metric and organizes peers in a unit size hypercube. However, the costs involved in the maintenance of the hypercube under churn are greater than the costs in a consistent hashing DHT, such as the proposed Hamming DHT. pSearch [Tang et al. 2003] is a P2P network that also uses content vectors and cosine similarity, but it differs from the Hamming DHT since it is specialized only for similarity search of text documents in a P2P network. Bhattacharya [Bhattacharya et al. 2005] uses co-

sine similarity and LSH functions to propose a framework for similarity search in P2P databases, such as Chord. Nevertheless, all these three mainly differ from the Hamming DHT because they do not explore the Hamming similarity of identifiers for organizing peers and contents in the ring and to facilitate the similarity search of any kind of content.

5.1. Organizing the Ring

In the Hamming DHT, the keys are mapped to peers in the network, and such keys can represent content identifiers or, depending on the application using the DHT, these keys can also be associated to values representing a locator or a list of locators of contents. Basically, each key/value pair is stored at the peer responsible for the slice that contains it, defined according to the Gray code sequence.

In order to operate, peers ($\mathcal{P}$) and contents ($\mathcal{C}$) in the Hamming DHT are mapped to the same identity space, using two different hash functions: ($\mathcal{F}_\mathcal{P}$ and $\mathcal{F}_\mathcal{C}$). In the case of the mapping regarding the peers, it is important that $\mathcal{F}_\mathcal{P}$ assures consistent hashing, since the ideal is that each peer becomes responsible for roughly a same size slice of the overall identity space. In this way, load balancing is provided and relatively little movement of keys occur when peers leave or join the DHT. Hence, the Hamming DHT adopts for $\mathcal{F}_\mathcal{P}$ a base hash function, like MD5 or SHA-1, and a peer identifier is obtained, for example, by hashing its name, IP address or a private key.

On the other hand, to map a content to the identity space, the Hamming DHT adopts for $\mathcal{F}_\mathcal{C}$ the Random Hyperplane Hashing function, previously described in Section 3, which is a non-uniform hashing function that meets the following properties:

$\forall\ c_1, c_2 \in \mathcal{C}$: *sim*(c_1,c_2) $\rightarrow$ [0..1], where 0 means no similarity.

$\forall\ c_1, c_2 \in \mathcal{C}$: D_h(*sim*($\mathcal{F}_\mathcal{C}(c_1)$,$\mathcal{F}_\mathcal{C}(c_2)$)) $\propto$ 1/*sim*(c_1, c_2),

where *sim* is any similarity function between contents, and D_h is the Hamming distance metric. In the Hamming DHT, a content identifier is obtained by hashing the content vector itself and the cosine similarity is used.

5.2. Consistent Hashing

After defining m-bit identifiers to all peers and contents using the mapping functions $\mathcal{F}_\mathcal{P}$ and $\mathcal{F}_\mathcal{C}$, the obtained identifiers are ordered according to the Gray code resulting in a ring of size 2^m. Figure 3 shows an example of the proposed Hamming DHT, where $m = 5$. As can be seen in this figure, there are fours peers (3 - 00011_2, 13 - 01101_2, 30 - 11110_2 and 22 - 10110_2), and three contents (0 - 00000_2, 24 - 11000_2 and 31 - 11111_2).

In order to store the contents in the Hamming DHT, the key *k* of a content is assigned to the first peer whose identifier is subsequent or equal to *k*. This peer is called the *successor* of key *k* and it is denoted as *successor(k)*. In short, the *successor(k)* is the first peer clockwise from *k* in the m-bit ring. For example, the peer 3 (00011_2) is responsible for storing the content of $k = 0$ (00000_2), the peer 30 (11110_2) is responsible for storing the content of $k = 24$ (11000_2), and the peer 22 (10110_2) is responsible for storing the content of $k = 31$ (11111_2).

The full list of identifiers according to the Gray code for $m = 5$ is presented in the right side of Figure 3, where the four peers are highlighted by the arrows. In a comparison between the Gray code sequence and the sequential binary (also shown in the right side of

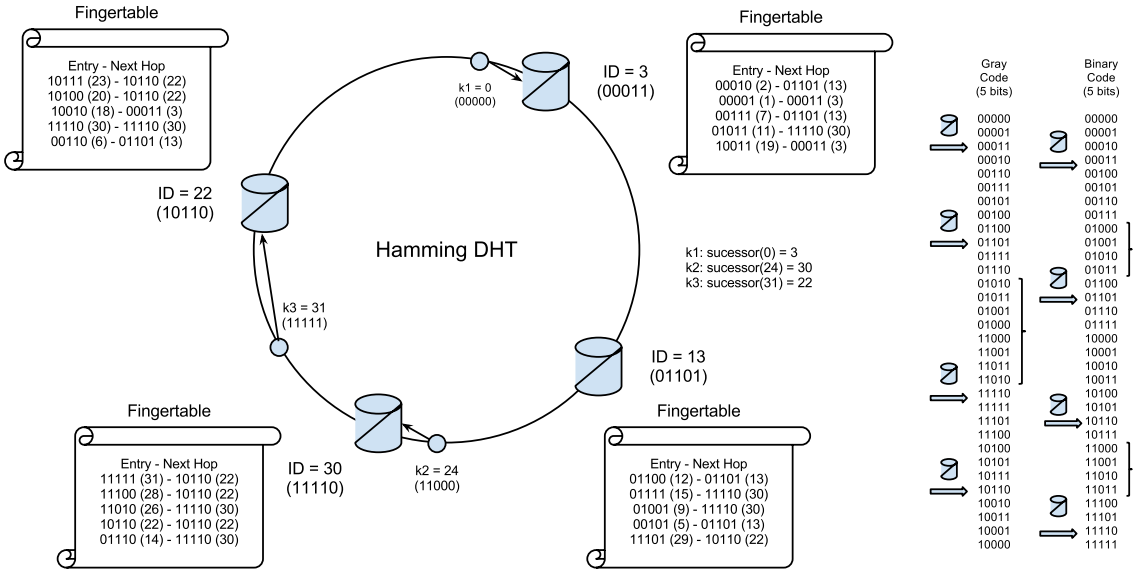


Figure 3. Example of the Hamming DHT ring with $m = 5$.

Figure 3), it is possible to realize the benefits of the Gray code sequence for aggregating similar content. As an example, the curly brackets show that, in a search for content identifiers similar to *10**, the use of Gray codes reduces the distance between them, as proven by Faloutsos in [Faloutsos 1988]. As can be seen in the figure, all the occurrences of *10** are consecutively positioned and stored in peer 30, while in the binary natural order, the occurrences are stored in peers 13 and 30.

In order to maintain the consistent hashing, whenever a peer $p \in \mathcal{P}$ joins the network, certain keys previously assigned to *successor(p)* now become assigned to *p*. When *p* leaves the network, all of its assigned keys are reassigned to *successor(p)*.

5.3. Establishing Fingers

Once the ring is organized, the storage and retrieval of information on/from the DHT is possible. To this aim, each peer must be aware of its successor on the ring and, based on this circular relationships, the actions of storing and retrieving a given key k simple require that the messages are routed around the ring, passing through the list of successor peers, until finding the peer responsible for that content key (the *successor(k)*).

However, to obtain a better routing performance, each peer maintains additional routing information about a few other peers. As in Chord, each peer maintains, in the steady state, information only about *O(log N)* other peers, where *N* is the number of peers in the network. In this schema, it is necessary to store information about only a small portion of the network. Afterwards, to allow contacting the peers present in the finger table, the Hamming DHT associates the locator (e.g., an IP address) of them in the respective entries. For the cases where the required finger entry does not correspond to a peer located in the DHT (it might be a content with key k), such entry is mapped to the *successor(k)*.

Each peer *p* maintains a routing table, also called finger table, with (at most) *m* entries. The i^{th} entry in the finger table (f_i) maps the identifier of the peer *p*, switching its i^{th} bit to its successor on the identifier ring, i.e., $f_i = (\mathcal{F}_{\mathcal{P}}(p) \oplus 2^{i-1}) \rightarrow sucessor(f_i)$,

$1 \leq i \leq m$. This finger table also includes the locator of *successor(f_i)*. As an example, in Figure 3 it is shown the finger tables of peers 3, 13, 30 and 22. In this example, the finger table of peer 13 (01101_2) points to the successor of each of its *m* entries: 12 (01100_2), which points to 13 (01101_2); 15 (01111_2), which points to 30 (11110_2); 9 (01001_2), which points to 30 (11110_2); 5 (00101_2), which points to 13 (01101_2); and 29 (11101_2) which points to 22 (10110_2).

For the cases when a peer p does not have a finger directly established with the *successor(k)* of key k, it forwards the packet to the peer p' available in its finger table, whose identifier of p' most immediately precedes k. Such process is repeated until it arrives at the *successor(k)*, corresponding to cases where the number of hops (the path) between the peers are bigger than one. This operation, as previously mentioned, is done according to the Gray code sequence.

Finally, in dynamic scenarios, peers can join and leave the network at any time. So, the ability of the system to locate every key in the network is necessary to be maintained. To achieve this goal, the correct maintenance of each peer's successor and the consistent hashing are necessary. To achieve a better performance, it is also necessary to maintain up-to-date the finger table of each peer. The algorithm to handle the peer joining and leaving in the system borrows the same core ideas from Chord, since the proposed Hamming DHT also deals with a ring, a consistent hashing, and adopts the same number of entries in the finger table of each peer.

The next section presents the evaluations performed by searching similar contents, indexed with the RHH function in the Hamming DHT and in Chord.

6. Evaluation

This section describes some experiments aiming to evaluate the Hamming DHT proposal. The main idea is to validate such DHT as a valuable approach in order to support the searching for similar contents. The following experiments were done:

- Sets with different quantities of content vectors have been generated with the cosine of the angle between them equal to or greater than 0.9, 0.8 and 0.5, respectively. It means that the contents in each set have similarity belonging to the following intervals: [0.9, 1.0], [0.8, 1.0] and [0.5, 1.0], respectively. For each set, using the RHH function presented in [de Paula et al. 2011], 128-bits and 64-bits content identifiers were generated;
- These sets of content identifiers were indexed and distributed at a time in the DHTs. The peers distribution was the same for each test in Chord and Hamming DHT. Ten different peers distribution (networks) were tested and evaluated for each DHT;
- For each set of similar contents, a query vector with the same similarity level of the respective set was generated;
- An identifier was generated for each query and a lookup having this identifier as argument was done in each DHT. The lookup message is forwarded to the peer which is responsible for the query identifier (the hosting peer), i.e., its successor on the ring;
- From the hosting peer, each content identifier agreeing with the query's similarity level is retrieved, and the distance in number of hops from the hosting peer to the other peers hosting each similar contents is measured.

These tests permitted to highlight the following points:

- The frequency distribution of the Hamming distances from the query originating host to each similar content in the set. This evaluation shows the effectiveness of the RHH function to generate content identifiers preserving in their Hamming distance the respective content similarities;
- The frequency distribution of the number of hops to retrieve all similar contents to the query in each set. This evaluation shows that the Hamming DHT aggregates more than a normal ring-style DHT, such as Chord, reducing the distance between similar contents in the number of hops;
- The query recall, corresponding to the fraction of the content that is relevant to the query and was successfully retrieved. This evaluation shows that it is possible to build a more efficient search engine on top of the Hamming DHT, at lower costs, measured in the number of hops to complete the query.

For the tests, a simulator of the Chord and the Hamming DHT was developed. This simulator is able to generate random peers and join them in a ring-style DHT. The algorithms presented in Section 4 were used to convert a binary natural ring to a ring following the sequence of the Gray code. Also, the developed simulator indexes and stores each content identifier *k* by the use of the *put(k,v)* operation. The *lookup(k)* operation returns the successor of the key *k* on the ring, which represents the peer responsible to store the content associated to this key, the hosting peer. The *get(k)* primitive was extended to handle the proposed similarity level, assuming the format *get(k,sim)*: given a key and a similarity level (*sim*), it returns all similar contents stored in the hosting peer. This search can be extended to the neighbors of the hosting peer with distance of 1 hop or longer distances, aiming to improve the searching results.

The tests were performed using two different key lengths, 64 and 128 bits, and the influence of this variable in the results was necessary to be evaluated. The frequency distribution was shown with a 64-bit key and the recall was shown with a 128-bit key. While not all results are presented here, there is no significant differences between them.

Figure 4 shows the frequency distribution of the Hamming distances, on average, from the query to each content identifier in its set, according to its similarity level. For an identifier with size equal to 128 bits, the results show that the RHH function is capable of maintaining the similarity level of each content identifier in their Hamming distances. From this figure it is possible to see that, with the similarity level equal to 0.9 and a 128-bits length identifier, most of the Hamming distances are less than 10 (7,81% of 128 bits) and the maximum Hamming distance is 13 (10,16% of 128 bits), corresponding to a Hamming similarity of 0,898 (89,8%). The Hamming similarity corresponds to the relation between the number of bits matching in two bit strings and the total size of these strings.

This test was repeated 10 times. The variance and 95% confidence interval of these results are negligible. It is important to explain here that the content vectors were generated having 100 dimensions, and the cosine of the angle between each content vector and the query vector having, respectively, values greater than or equal to 0.9, 0.8 and 0.5 (their similarity level).

Figures 5(a) and 5(b) show the frequency distribution of the number of hops to retrieve the similar contents. The figures exhibit the percentage of recovered contents,

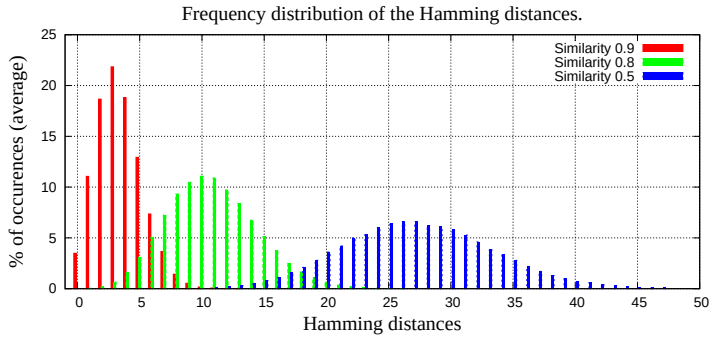


Figure 4. Frequency distribution of the Hamming distances from the query for the three content sets (Similarities 0.9, 0.8 and 0.5) using 128-bit keys.

which is relative to the total number of similar contents in the set, and their corresponding distance in to the hosting peer. As explained before, the query is forwarded to the hosting peer and, from this peer, look-ups for the totality of the similar contents in the set are performed. The tests were performed varying the number of peers in each DHT to analyze this influence in the results. The tests simulated 1000 (a) and 10000 (b) peers in Chord and in the Hamming DHT with similarity 0.5 (a) and 0.8 (b). The size of the keys used in these results is 64-bits.

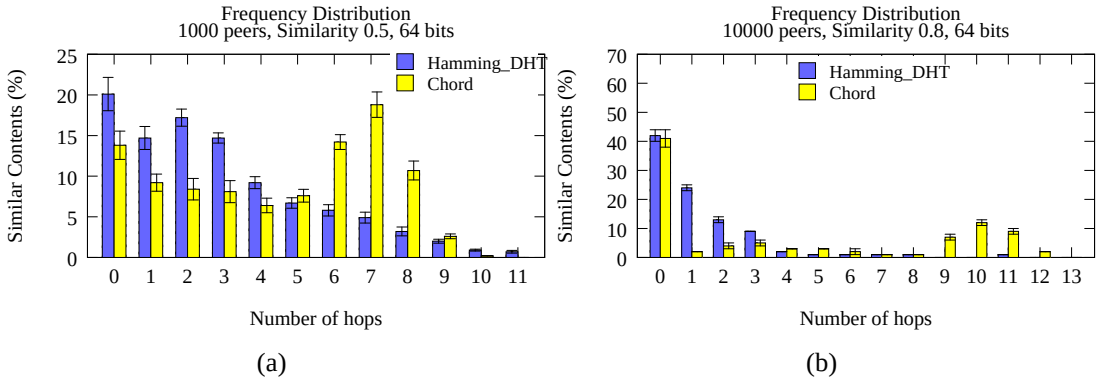
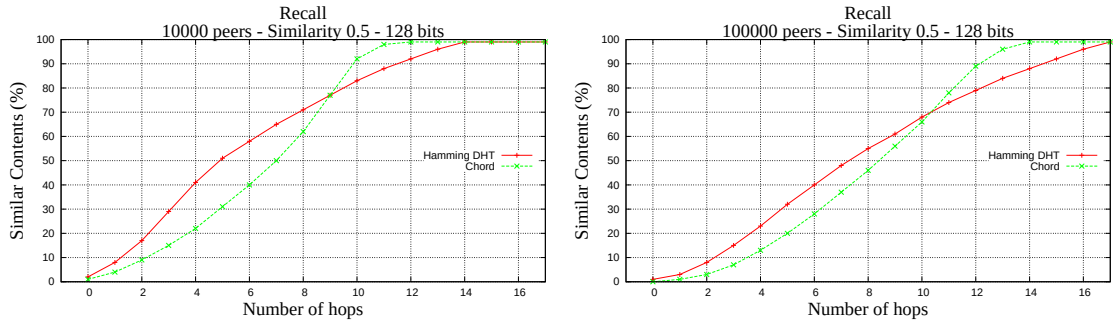


Figure 5. Distribution of the number of hops: 1000 peers, similarity 0.5 and 64-bit keys (a); 10000 peers, similarity 0.8 and 64-bit keys (b).

These results show the average obtained from 1000 simulations for each set of similar contents, which means that 100 different sets and queries were generated. Also, each of these 100 sets were tested in 10 different networks in Chord and the Hamming DHT. From these results, it is possible to notice that the Hamming DHT is able to cluster more similar contents in lower number of hops, i.e., lower distances between them. The confidence interval for 95% of the samples is shown in the vertical bars. The results hardly depends on the query identifier and the distribution of the peers in the DHT but, with this confidence interval, it is possible to say that, comparing to Chord, the Hamming DHT reduces the number of hops necessary to retrieve similar contents in a similarity search. The results for the similarity level equal to 0.9 are not shown due to space limitations, but it has an equivalent behavior to the similarity level 0.8.

Figure 5(b) shows that there is no significant difference between Chord and the Hamming DHT in the number of similar contents having a 0 hop distance from the hosting peer. This can be explained by the behavior of the RHH function under high similarity levels. As an example, suppose a 100-bit content identifier and a similarity level of 0.9 in a DHT having 1024 (2^{10}) peers. In this scenario, with consistent hashing in the peers' identifier, each peer is responsible for $2^{100}/2^{10} = 2^{90}$ identifiers. Two different content identifiers having similarity greater than or equal to 0.9 vary in up to 10 bits (with high probability). If this variation occurs in the 90 least significant bits, this variation implies that these content identifiers will be stored in the same hosting peer, consequently with 0 hop distance between them. Any variation in any of the 10 most significant bits, maps these similar content identifiers in different peers in the DHT, forcing one or more hops.

Figures 6, 7 and 8 show the recall in a similarity search for the Chord and the Hamming DHT. Figure 6(a) uses a DHT with 10000 peers, similarity level 0.5 and a 128-bit keys, while Figure 6(b) uses the same values for the similarity level and the length of the keys, but in a DHT with 100000 peers. Figure 7(a) uses a DHT with 10000 peers, similarity level 0.8 and a 128-bit keys, while in Figure 7(b) the DHT has 100000 peers. Figure 8(a) uses a DHT with 10000 peers, similarity level 0.9 and a 128-bit keys, while in Figure 8(b) the DHT has 100000 peers.



(a) (b)

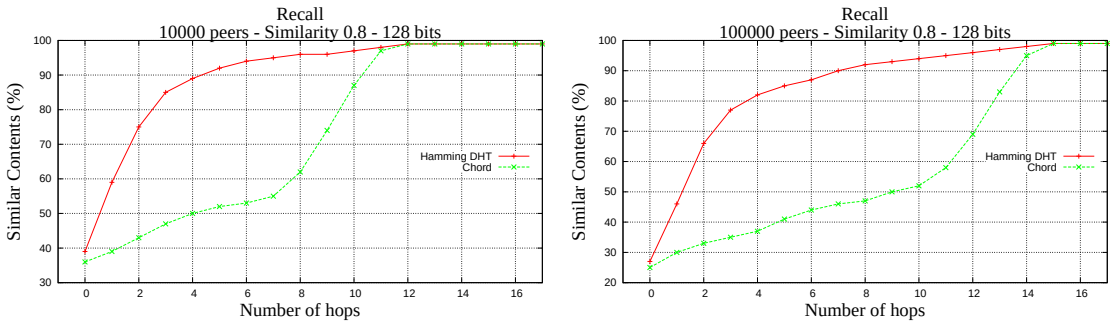
Figure 6. Recall, with 10000 peers (a) and 100000 peers (b), similarity level 0.5 and 128-bit key.

From the results of Figures 6, 7 and 8, it is possible to see that using the Hamming DHT as an infrastructure to support similarity search is a valuable approach. Because peers are distributed in an homogeneous way along the ring, each peer will be responsible for, approximately, the same number of keys in both DHTs, while the fingers acquisition according to the Hamming distance leads the Hamming DHT to benefit similar identifiers.

As an example, from Figure 7(a), it is possible to see that a search engine designed over the Hamming DHT having a *get(k,sim)* function with a depth of 4 hops, 90% of all similar concepts can be retrieved, while in Chord, only 50% of them can be retrieved.

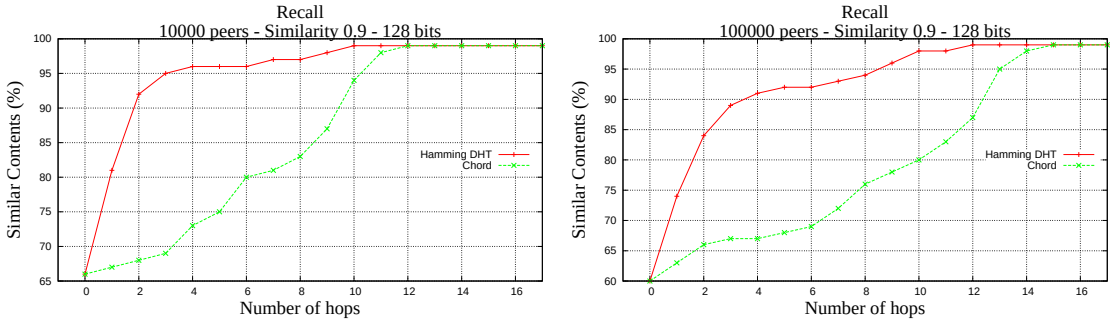
7. Conclusions and Future Work

This paper proposes a DHT based on the Gray code and the use of the Hamming distance as a similarity metric to facilitate the search for similar content, which we called Hamming



(a) (b)

Figure 7. Recall, with 10000 peers (a) and 100000 peers (b), similarity level 0.8 and 128-bit key.



(a) (b)

Figure 8. Recall, with 10000 peers (a) and 100000 peers (b), similarity level 0.9 and 128-bit key.

DHT. The tests done compared the average number of hops necessary to retrieve similar content on the Hamming DHT and in the Chord DHT, used as a reference. The idea was to experimentally evaluate the reduction in the distance to retrieve similar content, and the recall, that represents the relation between the number of retrieved content and the total number of similar content in the DHT with the lowest distance, measured in the number of hops from the query source.

The similar contents used in this paper are indexed using the RHH function and they are represented by content vectors. The similarity between contents is represented by the cosine of the angle of their content vectors and this similarity is maintained in the Hamming distance of their identifiers. The results show that the Hamming DHT can be a useful tool to serve as an infrastructure for similarity search. The comparison with Chord is not completely fair because it was not proposed in the context of similarity search. However, Chord can be used as a reference element to illustrate how a specific distributed structure, as the Hamming DHT, can contribute for this kind of searching. Also, to the best of our knowledge, no other paper in the DHT literature explores the Hamming similarity of content identifiers to propose a DHT specialized for similarity search, which difficults our comparisons with related work.

As future work, we intend to investigate a way to generate identifiers for the peers reflecting the similarity between them and the content they share. In this scenario, our hypothesis is that similar peers will be closer in their Hamming distance. Also, another scenario we intend to investigate is the use of this approach in a data storage data center.

References

Berry, M. W., Drmac, Z., Elizabeth, and Jessup, R. (1999). Matrices, vector spaces, and information retrieval. *SIAM Review*, 41:335–362.

Bhattacharya, I., Kashyap, S., and Parthasarathy, S. (2005). Similarity searching in peer-to-peer databases. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 329 –338.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, New York, NY, USA. ACM.

de Paula, L. B., Villaça, R. S., and Magalhães, M. F. (2011). Analysis of concept similarity methods applied to an lsh function. In *COMPSAC' 11: Computer Software and Applications Conference*, Munich, Germany. IEEE.

Faloutsos, C. (1988). Gray codes for partial match and range queries. *IEEE Trans. Software Eng.*, 14(10):1381–1393.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA. ACM.

Kong, F.-H. (2009). Image retrieval using both color and texture features. In *Machine Learning and Cybernetics, 2009 International Conference on*, volume 4, pages 2228 –2232.

March, V. and Teo, Y. M. (2005). Multi-attribute range queries in read-only dht. In *in Proc. of the 15th IEEE Conf. on Computer Communications and Networks (ICCCN 2006)*, pages 419–424.

Olszak, A. (2010). Hycube: a dht routing system based on a hierarchical hypercube geometry. In *Proc. of the 8th International Conference on Parallel Processing and Applied Mathematics*, PPAM'09, pages 260–269, Berlin, Heidelberg. Springer-Verlag.

Qian, G., Sural, S., Gu, Y., and Pramanik, S. (2004). Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *Proceedings of 2004 ACM Symposium on Applied Computing*, pages 1232–1237. ACM Press.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32.

Tang, C., Xu, Z., and Dwarkadas, S. (2003). Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 175–186, New York, NY, USA. ACM.