Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 31 (2008) 358–374
computer communications
www.elsevier.com/locate/comcom

Range queries over skip tree graphs

A. González-Beltrán *, P. Milligan, P. Sage

Queen's University Belfast, School of Electronics, Electrical Engineering and Computer Science, Belfast BT7 1NN, UK

Available online 14 August 2007

Abstract

The support for complex queries, such as range, prefix and aggregation queries, over structured peer-to-peer systems is currently an active and significant topic of research. This paper demonstrates how Skip Tree Graph, as a novel structure, presents an efficient solution to that problem area through provision of a distributed search tree functionality on decentralised and dynamic environments. Since Skip Tree Graph is based on skip trees, a concurrent approach to skip lists, it constitutes an augmentation of skip graphs that extends its functionality and allows for important performance improvements. This work presents a thorough comparison between these two related peer-to-peer overlay networks, their construction, search algorithms and properties. Being based on tree structures, skip tree graphs supports aggregation queries and multicast/broadcast operations, which cannot be directly implemented in its predecessor. The repair mechanism for healing the structure in case of failures is more efficient and harnesses the parallelism inherent in P2P networks. Particular consideration is given to the performance of different range-query schemes over the two related structures. Theoretical and experimental results conclude that Skip Tree Graphs outperform skip graphs on both exact-match and range searches.

Keywords: Structured peer-to-peer networks; Distributed search tree; Range queries; Aggregation queries; Concurrency

1. Introduction

Structured Peer-to-Peer (P2P) systems have built overlay networks by organising nodes in predefined topologies in the provision of large-scale, decentralised and dynamic routing platforms. Initial examples of structured P2P networks [1–4] provided a Distributed Hash Table (DHT) functionality. DHTs are based on consistent hashing [5] and, consequently, support efficient searches of keys identifying distributed resources while maintaining good load balancing properties. However, the main drawback of these systems is that the hashing approach destroys the semantic relation between the keys, and thus, there is no spatial locality among them. Consequently, DHTs do not directly support complex queries based on the key ordering, including range queries and nearest-neighbour queries. Moreover, DHTs assume a flat identifier space and thus, they do not directly support aggregation queries.

Dealing with complex queries efficiently is an important requirement for applications that could benefit from the scalability, fault-tolerance and dynamism of P2P systems [6,7]. Grid monitoring and information services and distributed data processing are some examples of these applications. A grid information service support queries about the capabilities and state of distributed resources available on a grid [8] and it should support range queries, prefix queries and aggregation queries. For instance, a user may want to discover all the computational resources whose CPU utilisation is less than 50% (expressed as a range query), find out the total load of a subset of computational resources or query about the free storage space on a subset of storage resources (aggregation queries summing up the corresponding attribute over a group or resources). Structured P2P systems have been proposed as an efficient routing platform for the provision of grid information services [9,10].

* Corresponding author. Tel.: +44 2031151143.
E-mail addresses: a.gonzalez-beltran@qub.ac.uk (A. González-Beltrán), p.milligan@qub.ac.uk (P. Milligan), p.sage@qub.ac.uk (P. Sage).

0140-3664/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2007.08.003

This paper seeks to augment the expressiveness of the query language for structured P2P systems through provision of support for order-based queries (range queries and prefix queries) and aggregation queries, given their importance for many applications. In addition, the network should offer efficient broadcast and multicast facilities, allowing the dissemination and collection of information on a global scale.

In order to meet this requirement, research to date has explored DHT-based solutions and overlay networks specifically built to provide a Distributed Search Tree functionality [11], i.e. the extension of DHT capabilities to support order-based and aggregation queries. DHT-based solutions can be classified into those which modify the DHT [9,6,12,13] and those which leave it unchanged [14,15]. The former group adapts the DHT so that complex queries can be performed. The latter group includes systems whose architecture has a DHT as a routing substrate component, on top of which another module dealing with complex queries is overlaid.

Systems not based on DHTs have designed overlay networks for which the order of the keys is preserved [16–18,11,19,20]. As hashing provides a solution to the dictionary problem, logical evolution has led to the consideration of other data structures that implement (ordered) dictionaries such as tries, 2–3 trees, B-trees, AVL trees, and skip lists.

Skip graph [17] is a structured P2P network that is based on skip lists [21]. A skip list is an increasingly sparse set of sorted doubly-linked list of keys, where the higher levels are used as shortcuts to reach nodes at greater distances quickly. The approach for the construction of skip graphs is to superimpose several skip lists, adding redundancy for the provision of fault-tolerance and the avoidance of congestion. Its search, insertion and delete operations are logarithmic in the number of nodes in the network.

This paper takes a further step by building a new P2P network denominated Skip tree graph, which is based on skip trees [22]. Skip trees are isomorphic to skip lists [21] while they provide a concurrent approach by incorporating in the structure information about the path followed by skip lists when performing sequential searches for keys [22]. They are suited for high degree of concurrency because their operations are local, i.e. access a small number of close nodes.

Skip tree graphs constitute an augmentation of the skip graph structure that keeps the good properties while extending its functionality with the support of aggregation queries and broadcast/multicast operations. Moreover, skip tree graphs provide a significant improvement in the performance on both exact-match and range searches. As a skip graph is an overlapping set of skip lists, skip tree graphs are defined as an overlapping set of transformed skip trees. Transformed skip trees combine skip lists and skip trees and their definition allows for an efficient construction of the distributed structure. Transformed skip trees incorporate what is defined as conjugate nodes, which constitute a shortcut in the vertical direction. Given that skip lists, skip trees and transformed skip trees are isomorphic (transformed skip trees are obtained as a sequence of reversible steps from skip trees), it results that skip tree graphs are isomorphic to skip graphs.

The skip tree graph structure was presented in a preliminary work [23] as the result of a set of transformations to related skip trees. This preliminary work focused on the analysis of exact-match queries. Two different exact-match query schemes over skip tree graphs were presented and compared with the skip graph search mechanism. These two schemes are [23]:

(1) a modification of the skip graph search by considering conjugate nodes while traversing the horizontal lists and
(2) a tree-based version that only considers conjugate nodes.

It was shown that, while these two new schemes improve skip graph search, the tree-based version outperforms the other two [23].

In this paper, the Skip tree graph structure is formally defined and its algorithms and its properties are presented in comparison with skip graphs. It is shown that the addition of conjugate nodes does not affect the expected logarithmic height of the structure and state of the nodes. It is proved that the exact-match search remains logarithmic on expectation, while the reduction of the constant of proportionality produces a significant improvement in its cost in terms of messages and hops. A more efficient maintenance mechanism than the one for skip graphs is presented, which repairs the structure in case of failures by taking advantage of the hierarchical structure.

In addition, the main focus of this work is related to the analysis of range queries over the two related skip-list-based structures. At the time of writing, the authors are not aware of other work performing such analysis. Two new range-query schemes are introduced, one for each of the structures. These are compared to the sequential and the broadcasting range-query approach [17]. Analytical and experimental results conclude that the range-query scheme over skip tree graphs outperforms the rest of the schemes. A discussion on how skip tree graphs support a greater degree of concurrency than skip graphs is also presented.

The organisation of this paper is as follows. Section 2 presents related work, i.e. solutions for range query support over structured P2P networks. Section 3 formally defines the skip tree graph structure, analyses its properties, presents an analysis on the tree-based search operation, describes the insert operation in detail together with its performance analysis and introduces the efficient repair mechanism for skip tree graphs. In Section 4 different schemes for range queries over skip graph and skip tree graph are analysed, providing proofs of their performance costs. Next, experimental results are presented that confirm that

the skip tree graph range query scheme outperforms the rest of the schemes. Finally, the last section presents the conclusions.

2. Related work

The provision of support for range queries over DHTs has considered two approaches: either adapting the DHT to facilitate richer query functionality or using it as a building block in a layered architecture.

Since the feature that prevents DHTs directly supporting range queries is the use of hashing to map the data in the distributed nodes, modifications have used different functions to do this mapping, while pursuing the preservation of the order among the keys or the relation between the ranges. Andrzejak and Xu [9] propose one solution that uses a space-filling curve (SFC) over CAN. Gupta et al. [12] use locality sensitive hashing over Chord, satisfying that similar ranges are hashed to the same peer with high probability. However, their solution only allows the computation of approximate answers for range queries. Sahin et al. [24] improved the previous approach by allowing exact range-query solutions. Pitoura et al. [13] attack simultaneously the problems of providing efficient range query processing and data-access load balance. Their solution uses a hash function which is both locality-preserving and randomised.

Using a DHT as an application-independent routing substrate gains on simplicity of implementation and deployment of P2P applications [15]. Systems following this approach rely on the DHT for scalability and fault-tolerance properties. In addition, they need to overlay other components on top of the DHT, whose interface provides only put/get operations, to deal with the requirements for richer semantics. P-Tree [14] uses Chord as the routing architecture and builds a B^+-tree on top. Each P2P node represents a leaf node in a B^+-tree and maintains the path from the index root to the leaf node. Chawathe et al. [15] designed a Prefix Hash Tree (PHT) which is a trie-based structure layered on top of a DHT, allowing for one-dimensional range queries. To perform a range query, the PHT has to perform multiple DHT-lookup queries and the query-cost is data dependent. While this approach is effective in reducing implementation and deployment complexity because it can outsource the DHT functionality, this is achieved at the expense of performance [15].

Non-DHT based approaches have considered solutions for ordered dictionaries in the sequential domain (such as tries, 2–3 trees, skip lists, AVL-trees) and have built overlay networks where the connections among peers are influenced by these data structures.

P-Grid [16] is based on the trie structure which is induced by recursively bisecting the data space. Peers are associated with each partition generated and have random connections to other peers such that prefix routing is enabled. Two approaches for range queries over P-Grid have been introduced [25]: sequential and parallel. The cost of the parallel approach depends on the distribution of the data and the size of the answer set but it is independent of the size of the range.

BATON [19] is a DST based on an AVL-tree augmented with extra connections at each level for fault-tolerance and load balancing. BATON* [20] extends the idea to a multi-way-tree and speeds up the searches by increasing the fan-out. Both structures undertake rigid insertion procedures, where a restructuring process affecting several nodes is needed. This restructuring process is expensive and prevents concurrent insertions.

Skip graphs [17] and SkipNet [18] are randomised structures based on skip lists and achieve expected $O(\log n)$ query and update times with $O(\log n)$ node state in an n-nodes network. Skip graphs preserve the order of the keys but do not provide load balancing. Alternatively, providing load balancing would prevent preserving the order of the keys [17]. A bucketing scheme [26] has been proposed to overcome this drawback. Skip B-trees [11] also extend skip graphs by combining them with B-trees. Their focus is on reducing nodes' state and providing a policy for assigning resources to nodes. This policy allows the achievement of good load balancing.

Skip tree graph is another extension of skip graphs, by means of a skip tree. Skip tree graphs inherit the same problem as skip graph with respect to load balancing. Approaches similar to the ones designed to improve skip graphs [26,11] could be adapted for skip tree graph. Being based on skip trees, which are a concurrent approach to skip lists, it requires only local operations and provides a greater degree of concurrency than other balanced-tree based approaches.

When classifying balanced search trees according to the set of rules they use, two groups are determined: split-and-join based and rotations based [22]. B-trees, skip lists and skip trees are in the first group and AVL-trees are in the second group. Therefore, skip graphs [17], skip B-trees [11] and skip tree graphs are P2P versions lying on the first group, while BATON [19] and BATON* [20] can be included in the second group.

3. Skip tree graph

In order to formally define *skip tree graphs*, a notation based on the notation used in [17] will be introduced next. Later, an overview of the related randomised structures *skip lists*, *skip trees*, *transformed skip trees* and *skip graphs* is given.

Let $(K, <_K)$ be a totally ordered domain, the set of keys stored in the structures. Let Σ be a finite alphabet such that a word from this alphabet denotes the random choices made for each key in the probabilistic data structure being considered. This word has been denominated *membership vector* [17]. Let $|\Sigma|$ be the cardinality of the alphabet and assuming that all the symbols in the alphabet are equally probable, each symbol occurs with probability $p = |\Sigma|^{-1}$. Let Σ^* be the set of all finite words consisting of characters in Σ, and let Σ^∞ consist of all infinite words. The membership vector of key k, denoted by $m(k)$, is potentially infinite ($m(k) \in \Sigma^\infty$). However, only a $O(\log n)$-prefix of $m(k)$ needs

to be generated on average for the keys in all the data structures. Given a word $w \in \Sigma^{\infty}$, its length is denoted $|w|$, its symbol at position i is w_i, its prefix of length $i \in \mathbb{N}_0$ is denoted $w|_i$ and $w|_0 = \epsilon$ is the empty word. Given two words w_1, $w_2 \in \Sigma^{\infty}$, their longest common prefix $lcp(w_1, w_2)$ is a word w such that $\exists i |w_1|_i = w_2|_i = w$ and $\nexists j > i$ such that $w_1|_j = w_2|_j$. Finally, P_w is defined as the set of all possible prefixes of w, i.e. $P_w = \{z \in \Sigma^*: \forall i \geqslant 0,\ z = w|_i\}$.

A *skip list* [21] is a random data structure introduced by Pugh, which consists of a family of sorted linked lists $\{L_\ell\}_{\ell \geqslant 0}$ (see Fig. 1a). The bottom list contains all the inserted keys and subsequent levels include random subsets of the keys from the immediate lower level, providing shortcuts to distant keys. A key k belongs to level ℓ if $m(k)|_\ell = w|_\ell$, for a given word $w \in \Sigma^{\infty}$. This word w identifies the skip list, which is denoted by $SL_{w,\Sigma}$ or simply SL_w when it is clear which is the alphabet used. In the special case $\Sigma = \{0,1\}$ and $w = 1^{\infty}$, the condition for k to belong to level ℓ is $m(k)|_\ell = 1^\ell$, which may be interpreted to mean that a key belongs to level ℓ if ℓ successive tosses of a coin return heads. Fig. 1a presents the skip list $SL_{011,\{0,1\}} = SL_{011}$. In general, for $\ell \geqslant 0$, $L_\ell = L_{w|_\ell} \supset L_{\ell+1} = L_{w|_{\ell+1}}$ and $SL_w = \{L_z\}_{z \in P_w}$.

A *skip tree* [22], introduced by Messeguer, is a structurally equivalent data structure to a skip list, i.e. there is a one-to-one mapping between the two structures. Fig. 1b shows an example of a skip tree equivalent to the skip list of Fig. 1a. A skip tree is derived from the path followed by the sequential search algorithm on skip lists [22], by adding information about this path in the nodes themselves. This addition allows for the implementation of concurrent algorithms based on local rules [22]. In a skip tree identified with word w, denoted by ST_w, a key k belongs to level i if and only if $lcp(m(k), w) = i$. Consecutive keys with the same level are grouped into a single node while the search tree property is maintained. 'White' or empty nodes are added so that all leaves in the tree are at the same level. Thus, the skip tree is considered as an *unbounded random B-tree* [22]. A skip tree is similar to a B-tree because all leaves are at the same level, nodes have a set of keys and pointers to children nodes and the search tree property is satisfied. However, it is *unbounded* and *random* because the number of keys in a node is not fixed as in the B-tree, but follows a probability distribution.

A *transformed skip tree* ST'_w (see Fig. 1c) combines a skip list and skip tree. A key k belongs to all the levels $\ell \geqslant 0$ such that $m(k)|_\ell = w|_\ell$. The maximum level is $\mathcal{L}$ if and only if $|lcp(k,w)| = \mathcal{L}$ and this level contains a key identified as the root, whose membership vector is $lcp(k,w)$. ST'_w is equivalent to SL_w with the addition of *conjugate keys* (note that to identify a root, it might be necessary to expand the membership vector of the keys at SL_w's top level to distinguish keys and conjugate keys). Two keys are said to be *conjugate* if their membership vectors share the same prefix of length $\ell - 1$ and differ in the symbol at position ℓ. In symbols, a *conjugate at level* ℓ, for all $\ell \geqslant 1$, for key k_1, denoted ℓ-*conjugate*(k_1) is k_2 such that $m(k_1)|_{\ell-1} = m(k_2)|_{\ell-1}$ and $m(k_1)|_\ell \neq m(k_2)|_\ell$. Thus, a key k appears as conjugate in ST'_w at level ℓ if its maximum level is $\ell - 1$, i.e. $|lcp(w, m(k))| = \ell - 1$. A node v stores a (potentially empty) ordered list of conjugate keys at each level, denoted $C_\ell(v)$, corresponding to the conjugate keys at level $\ell - 1$ between itself and its left neighbour at level ℓ. Conjugate nodes are indicated with smaller squares than nodes in Fig. 1c. As an example, $C_\ell(80) = [60, 70]$. The conjugate keys correspond to the nodes at each level of the corresponding skip tree ST_w. Note that the membership vectors corresponding to nodes 50 and 80 in Fig. 1c have been expanded to distinguish between a root and its conjugate.

Aspnes and Shah [17] presented a generalisation of a skip list called *skip graph*, in which redundancy is incorporated to allow the system to work efficiently in a dynamic

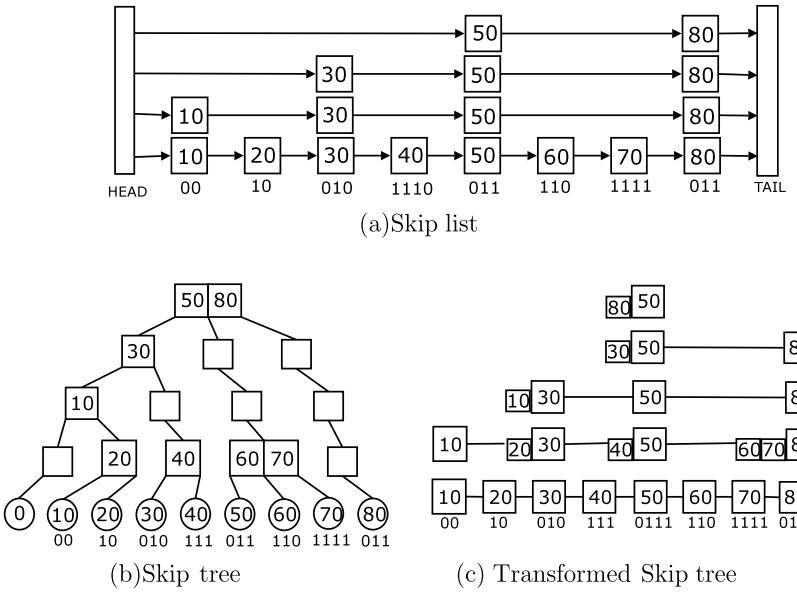

Fig. 1. Three structurally equivalent data structures: (a) skip list SL_{011}, (b) skip tree ST_{011}, (c) transformed skip tree ST'_{011}.

distributed environment such as the one assumed for P2P systems. The skip graph structure [17] is defined as a family $\{L_z\}_{z\in\Sigma^*}$ of sorted doubly-linked lists, where L_z contains all k for which z is a prefix of $m(k)$ for all $z \in \Sigma^*$. A particular L_z is part of level i if $|w| = i$. Thus, as for every $w \in \Sigma^\infty$, the family $\{L_z\}_{z\in P_w}$ is a skip list, a skip graph is a family of skip lists $\{SL_w\}_{w\in\Sigma^\infty}$, which overlap in the lower levels.

Finally, a *skip tree graph* is a family of transformed skip trees $\{ST'_w\}_{w\in\Sigma^\infty}$, which overlap in the lower levels. Fig. 2 provides an example of a skip tree graph. Eliminating the conjugate nodes, its equivalent skip graph is obtained. The design of the transformed skip tree aims to maintain the skip graph property that level $\ell+1$ can be built through local operations on level ℓ. This is achieved by maintaining the neighbouring connections at each level and it is required for the insertion operation.

The performance analysis for skip lists, skip trees and skip graphs has shown that insertion, deletion and searching operations take expected $O(\log n)$ time [21,22,17].

In the case of skip tree graphs, the same expected upper bound is maintained for its operations. As a skip tree graph is a super set of a skip graph, skip graph search operations could be performed over skip tree graphs. The addition of conjugate keys allows tree-based operations, which shorten the search paths in the vertical sense. This paper shows that by the augmentation of a skip graph with conjugate nodes, the asymptotic logarithmic behaviour of exact-match is not affected and in fact, its cost is improved by reducing the constant of proportionality of the upper bound. A range query scheme for skip tree graphs is also shown to outperform other schemes for skip graphs. To achieve this, the insertion procedure needs to add conjugate nodes. Although the insertion cost increases with respect to skip graph's cost, it is shown that it remains logarithmic in the number of nodes. The value of the increment depends on the cardinality of the alphabet Σ.

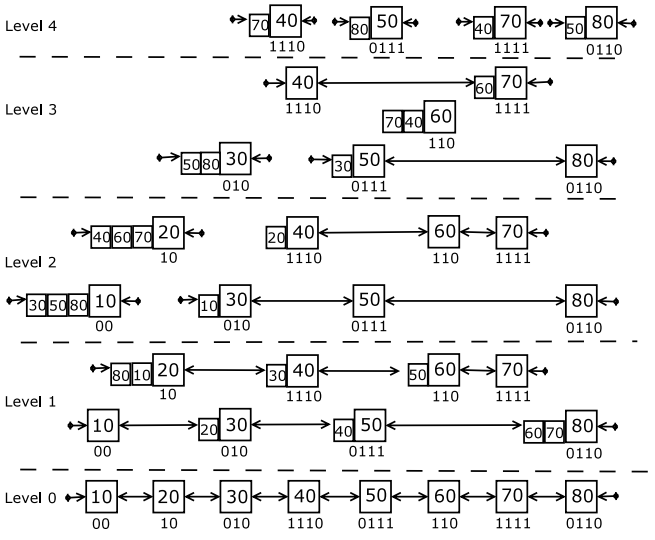

Fig. 2. A skip tree graph.

Apart from providing performance improvements in the search operations, skip tree graphs extend skip graphs' functionality. While skip graphs support range and prefix queries, they do not support aggregation queries. Aggregation queries combine the results on a large number of nodes, and thus, invariably rely on hierarchical structures to perform this combination of results. Being based on tree-structures, skip tree graphs allow for the implementation of aggregation queries. Nodes at each level constitute an aggregation point of all its conjugate nodes at that level. Some examples of aggregation functions are count, sum, maximum, minimum, average, and median. Similarly, the implementation of broadcasting or multicasting schemes over skip tree graphs could rely upon the underlying hierarchical structures.

3.1. Skip tree graph properties

The expected height of a skip tree graph coincides with the average height of the equivalent skip graph and is $E[H] = O(\log_{1/p} n)$ [17].

The number of conjugate nodes for a particular node v at level ℓ in a skip tree graph is a random variable $G_{v,\ell}$. In terms of the membership vectors, $G_{v,\ell}$ is the number of keys at level $\ell-1$ between v and its left neighbour at level ℓ, denoted vL_ℓ. Then, $m(v)|_\ell = m(vL_\ell|_\ell)$. Consequently, $G_{v,\ell}$ counts the number of nodes w to the left of v, at level $\ell-1$ such that $m(w)_\ell \neq m(v)_\ell$ until vL_ℓ is found. Thus, $G_{v,\ell}$ is a geometric random variable with probability of success p, $Pr[G_{v,\ell} = i] = q^i p$ and $E[G_{v,\ell}] = \frac{q}{p}$. This magnitude is called length of an internal node in a skip tree [22].

The state of a node in a skip tree graph includes two neighbours at each level ($O(\log_{1/p} n)$) plus a set of conjugates, whose cardinality is qp^{-1} on average. Thus, it is $O((2+qp^{-1})\log_{1/p} n)$. This implies that for a fixed p, the state remains logarithmic on average. The benefits offered by the skip tree graph structure come at the expense of requiring greater state. The increment with respect to the state of the skip graph is $O(qp^{-1}\log_{1/p} n)$.

3.2. Exact-match search operation

Taking advantage of conjugate nodes allows the improvement of the exact-match search cost either by accelerating horizontal traversals in skip graph search or by following the underlying tree-structures. Experimental results comparing these schemes with the skip graph scheme were presented elsewhere [23]. Next, the tree-based scheme is described and analytically compared with the skip graph scheme. A tight bound for the cost is given in order to show that the constant of proportionality is less than the corresponding constant for the exact-match search in skip graphs.

A *tree-based exact-match search operation* over a skip tree graph starts at an arbitrary node v and it traverses the underlying skip tree for which v is the root node. Thus, only conjugate nodes at each level need to be considered

and the search works as in a search tree by selecting the most appropriate conjugate node at each level or going down a level in the same node if there is no such conjugate. Intuitively, the search is bounded by the height of the skip tree and then, it takes expected $O(\log_{1/p} n)$ hops and messages.

Pugh showed an upper bound on the cost of the search for skip lists by analysing the search in the backward direction [21]. The procedure involves obtaining an expression for the cost of the search in an infinite list, using it to bound the cost to get up to level $L(n) = \log_{1/p} n$ and then, adding a bound for the cost from level $L(n)$ to the expected maximum level [21]. The expression for C_k, the expected cost of the search that climbs up to k levels, is obtained by considering that being at node v and level ℓ the two possible options are: to go up to level $\ell + 1$ while remaining in the same node (whose probability is p, the probability that node v is at level $\ell + 1$) and to go back to the left neighbour in the same level (with probability $1 - p$) [21]. Consequently, C_k follows the recurrence relation

$$\begin{cases} C_0 = 0 \\ C_k = (1-p)(1+C_k) + pC_{k-1} \end{cases}$$

whose solution is $C_k = \frac{1-p}{p}k$. Then, the search cost for a skip list is $S_n^{SL} \leqslant \frac{1-p}{p}\log_{1/p} + 1 + \frac{1}{1-p}$.

Note that the recurrence relation used here is not exactly the same as in [21]: the model assumed in this paper is that while each message takes at most unit time (one hop) to be delivered, internal processing at a peer takes no time. Therefore, the only operation that increases the cost is to go back to other node. Another important observation is that this is a tight upper bound [27] and by showing a lower upper bound for the search in a skip tree, the difference in the bounds will reflect the improvement properly.

When applying the same reasoning for the search over skip trees, the two options in the backwards direction are: to follow a conjugate "pointer" up (with probability $1 - p$) or to go up a level within the same node (with probability p). Then, the recurrence relation is:

$$\begin{cases} C_0 = 0 \\ C_k = pC_{k-1} + (1-p)(1+C_{k-1}) \end{cases}$$

whose solution is $C_k = (1-p)k$. It results that the search cost for a skip tree can be bounded by the expression: $S_n^{ST} \leqslant (1-p)(\log_{1/p} n) + \frac{1}{1-p}$.

Observing the constants accompanying the logarithms, it results that $\frac{1-p}{p} > 1 - p$. The bound for the search in skip tree graph is $\frac{1}{p}$ times less than the bound for the search in skip graphs. As the bounds are tight, the search in the skip tree graph reduces the cost of the skip graph's search. For instance, when $p = \frac{1}{2}$, there is a reduction of 50% in the cost.

3.3. Insert operation

In order to build an skip tree graph the insertion operation has to determine neighbours and conjugate links for the joining node. Establishing neighbour nodes at each level is performed similarly to the insert operation of skip graph [17]. However, this insertion procedure needs to be augmented to consider conjugate nodes.

A new node that wants to join the network needs to know a node already in the network. The new node v will insert itself in one linked list (as a neighbour node) at each level until it is in a singleton list at the topmost level. In addition, it has to be added as a conjugate node to the closest nodes to its right appearing in the rest of the lists at each level. In total, when searching to the right, $|\Sigma| = p^{-1}$ nodes need to be found. The new node also stores a set of conjugates at each level that are the nodes traversed when searching for the appropriate left neighbour for the next level.

While the skip graph operation consists of two stages [17], the skip tree graph insertion consists of the same two stages augmented to consider conjugate nodes and it might require a third stage if the new node was not added as conjugate in all the required lists during the second stage. The stages are:

(1) The new node v searches for its key in the network, finding the neighbours at the bottom level, and connects to them.
(2) For each level $\ell \geqslant 0$, the new node v searches for the closest nodes u and w at level ℓ such that $u <_K v <_K w$ and $m(v)|_{\ell+1} = m(u)|_{\ell+1} = m(w)|_{\ell+1}$. The nodes u and w are the left and right neighbours for level $\ell + 1$, respectively. While traversing the list at level ℓ to find u and w, the path is stored in respective lists c_v and c_w, where c_v contains the conjugate nodes of v and c_w the conjugate nodes of w. If the list c_w is not empty, v is added as conjugate to each node $t \in c_w$ such that the symbol $m(t)_{\ell+1}$ from Σ appears for the first time in the traversal.
(3) If c_w is empty (i.e. there is no conjugate node between v and w) or not all symbols from Σ are represented, then w forwards a message `findConjugatesOp` to its right neighbour searching for conjugates for v for the remaining symbols, if they exist.

This procedure makes evident the need to keep neighbouring links at all levels to build the structure.

Before every new node v connects to any neighbour, it is verified that no other nodes have joined in between. If there is a node that joined in between, the order of the nodes will be violated. Thus, when an existing node receives a message to link to the a new node v, it checks that connecting to v will maintain the proper order of the keys. If linking to the new node would result in violating the key-ordering, the old node transfers the message to its appropriate neighbour. This procedure ensures that a new node links to correct nodes at each level, even in the case of nodes concurrently joining the P2P network.

Once a new node is connected to a neighbour, these two adjacent nodes may need to split and re-allocate their

conjugate nodes. For an example of the insert operation see Fig. 3.

Summing up, the insertion procedure for node u traverses each list ℓ to the right to find the right neighbour and p^{-1} conjugate nodes that will link to u at level $\ell+1$, if they exist. The expected bound for the number of messages and hops for this operation is expressed in Lemma 1, whose proof is detailed in Appendix A. The list at level ℓ is also traversed to the left to find the left neighbour. The new node u includes in its conjugate table all the nodes found in the path from u to its left neighbour.

Lemma 1. *The expected number of messages and hops needed for u to find its right neighbour and one conjugate for each symbol in* $\Sigma - \{m(u)_{\ell+1}\}$ *at level* ℓ *is less than* p^{-2}.

Theorem 2. *The insert operation in a skip tree graph with n nodes takes expected* $O(\log n)$ *messages and hops.*

Proof of Theorem 2 Neighbours in the bottom level are found after a search operation, which takes $O((1-p)\log_{1/p} n)$ hops and messages on average. At each level $\ell \geqslant 1$, the new node u communicates with $\frac{1}{p}$ nodes on average to find its left neighbours v satisfying $m(u)|_{\ell+1} = m(v)|_{\ell+1}$ (taking $O(\frac{1}{p}\log_{1/p} n)$ hops and messages on average). In the right direction, u needs to search for its right neighbour w, verifying $m(u)|_\ell = m(v)|_\ell$ but also a set of conjugates c_σ such that $m(c_\sigma)_{\ell+1} = \sigma$ for $\sigma \in \Sigma - \{m(u)_{\ell+1}\}$, i.e. one conjugate for each symbol σ different to $m(u)_{\ell+1}$. By Lemma 1, the expected number of hops and messages in the right direction is bounded by p^{-2} at each level. Consequently, the total cost for the insert operation in skip tree graphs is $O((1-p+p^{-2}+p^{-1})\log_{1/p} n)$ that for a fixed p is $O(\log n)$ hops and messages on average. □

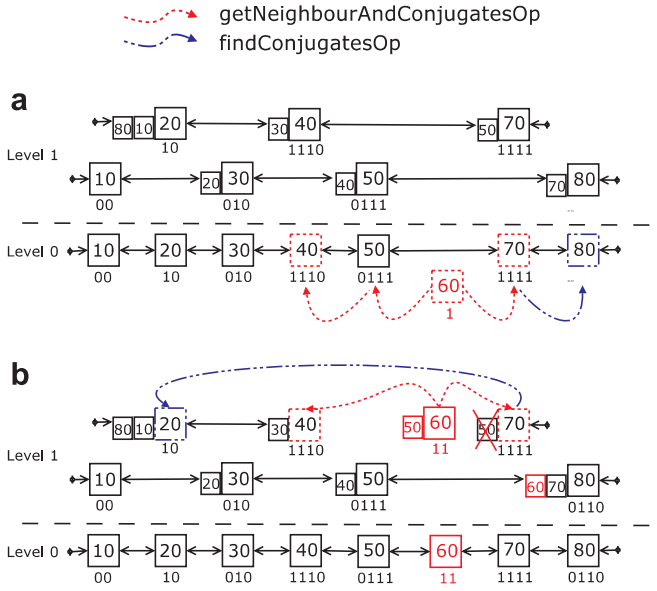

Fig. 3. Example of insert operation. (a) Node 60 inserts itself at the bottom level and searches for the appropriate neighbours for the first level in both directions (sending messages `getNeighbourAndConjugatesOp`). After finding the right neighbour at level one, it has not been inserted as a conjugate node. Thus, its right neighbour issues a message `findConjugatesOp` to find 80. (b) Node 60 inserts itself at level 1 where 70 splits its conjugates (and 50 becomes a conjugate for 60). Then, node 60 searches for its appropriate neighbours for the following level resulting in its adjacent neighbours. As 60 has not been added as conjugate after finding the right neighbour, 70 sends a `findConjugatesOp` to its right, finding 20.

In comparison with the insert operation for skip graphs, it is observed that while finding the appropriate location at the bottom level has better performance for skip tree graphs, they incur an extra cost when determining conjugate nodes. When the number of elements in the alphabet Σ increases, the difference on the cost increases. Then, the improvements on exact-match and range queries and added functionality are at the expense of a more costly insert operation. Experimental results on the cost of the insertion are given in [23].

3.4. Maintenance algorithm

In the presence of node or link failures, a repair mechanism or maintenance algorithm is required to reorganise the structure, healing the disruptions and avoiding performance degradation.

Skip graph's repair mechanism [17] is based on verifying a set of constraints that guarantee that the structure is correct. These constraints include [17]:

(1) key-order constraints: a key is greater than its left neighbour and less than its right neighbour.
(2) back-pointer constraints: ensuring that each list at each level forms a proper doubly-linked list.
(3) inter-level constraints: guaranteeing that neighbours at level ℓ are the closest neighbours at level $\ell-1$ whose membership vectors' ℓ-prefixes coincide with the node's membership vector ℓ-prefix.

Key-order constraints are invariants in any execution of the skip graph. Thus, the repair mechanism implements two operations for repairing invalid back-pointer and inter-level constraints, respectively. Checking inter-level constraints requires the traversal of the immediate lower level list and action upon the possible cases violating the constraint. The repair mechanism requires time quadratic in the number of nodes in the worst case.

A graph is a skip tree graph if, in addition to the previous constraints, it satisfies the following properties defining conjugate nodes:

(1) If $xL_\ell \neq \perp$, then $\exists k$ such that $xL_\ell = xL_{\ell-1}^{k+1}$ and $\forall j | 1 < j < k+1$ $m(x)|_\ell \neq m(xL_{\ell-1}^j)|_\ell$, $xC_\ell^j = xL_{\ell-1}^j$.
(2) If $xL_\ell = \perp$, then $\forall j | m(x)|_\ell \neq m(xL_{\ell-1}^j)|_\ell$, $xC_\ell^j = xL_{\ell-1}^j$.

For skip tree graph, it is possible to extend skip graph's back-pointer and inter-level repair operations to record the conjugate nodes and split them when setting new neighbours, if necessary. The split operation is as in the insertion

algorithm. However, by exploiting the hierarchical structure, it is possible to increase the parallelism of the operation that checks inter-level constraints. Instead of traversing the immediate lower level to verify proper assignment of neighbours, a node could use the information on its conjugate nodes to check simultaneously inter-level and conjugates constraints. In this way, a more efficient repair mechanism is obtained as follows.

For each level $\ell > 0$, each node x sends messages in parallel to all its conjugate nodes, if they exist. Assuming that $xC_\ell \neq \emptyset$, $|xC_\ell| = k+1$ and $\forall i | 1 \leqslant i \leqslant k, xC_\ell^i \neq \perp$, these messages aim to check the following conditions (which arise from the aforementioned constraints on conjugate nodes):

(1) the left neighbour of x at level $\ell-1$ is the rightmost conjugate at level ℓ, i.e. $xL_{\ell-1} = xC_\ell^1$.
(2) the left neighbour at level $\ell-1$ of each conjugate is the following conjugate to the left, in symbols: $xC_\ell^i L_{\ell-1} = xC_\ell^{i+1}$
(3) the left neighbour at level $\ell-1$ of the leftmost conjugate is the left neighbour of x at level ℓ, i.e. $xC_\ell^k L_{\ell-1} = xL_\ell$.

Without loss of generality, each conjugate is considered not $\perp$, as if any of the conjugate nodes is $\perp$, it can be eliminated and the same conditions as before are checked. On the other hand, if $xC_\ell = \emptyset$, then the procedure checks if $xL_\ell = xL_{\ell-1}$.

Thus, each conjugate xC_ℓ^i receives a message `checkLeftNeighbourOp` $(\ell-1, y, x)$ from x, and checks if its left neighbour at level $\ell-1$ is the node y as indicated by x (see Fig. 4 for an example of how the connections are established at levels ℓ and $\ell-1$). If the neighbour is correct, the constraint is not violated and no repair action is required. If this is not the case and xC_ℓ^i's left neighbour is z, one of the following cases violating the constraints occur (the cases are outlined in Figs. 5–7, respectively):

(1) $z = \perp$
(2) $z \neq \perp \wedge m(z)|_\ell = m(x)|_\ell$
(3) $z \neq \perp \wedge m(z)|_\ell \neq m(x)|_\ell$

In the first case, xC_ℓ^i sets y, the node suggested by x, as its left neighbour in level $\ell-1$ and sends a message to y. In the same way as in the insert operation, y verifies that linking to xC_ℓ^i will preserve the ordering constraints. If this is not the case, y finds the appropriate neighbour for xC_ℓ^i.

In the second and third cases, xC_ℓ^i sends a message back to x letting it know about the existence of its left neighbour z. If $m(z)|_\ell = m(x)|_\ell$, x will set z as its new left neighbour at level ℓ, splitting its conjugate nodes and sending a message to z. In turn, z confirms that x will be its new right neighbour or finds an appropriate node in its neighbourhood to link to x.

Otherwise, when $m(z)|_\ell \neq m(x)|_\ell$, z (and possibly some of its neighbours) will be inserted as a new conjugate between xC_ℓ^i and xC_ℓ^{i+1}. Additionally, x will send a `checkLeftNeighbourOp` $(\ell-1, xC_\ell^{i+1}, x)$ message to z.

Then, the skip tree graph repair mechanism involves two operations: the back-pointer repair (as in skip graphs) and parallel inter-level repair using conjugates, as explained above.

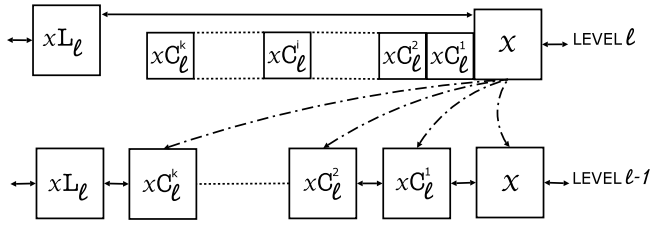

Fig. 4. Repair mechanism: node x checks that xC_ℓ^1 is its neighbour at level $\ell-1$ and sends `checkLeftNeighbourOp` messages to all its conjugate nodes.

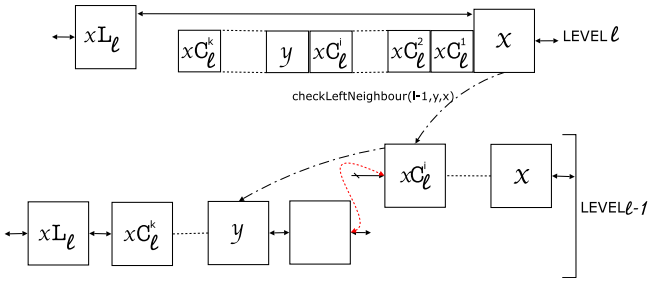

Fig. 5. Parallel inter-level repair: case $z = \perp$.

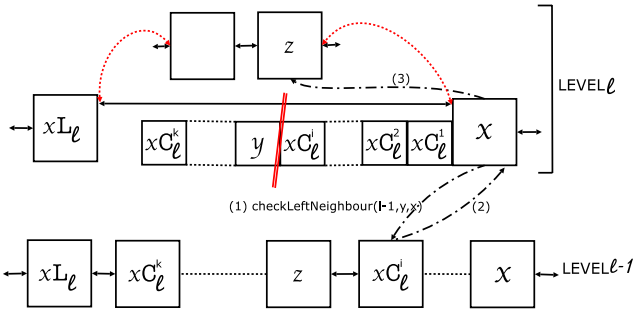

Fig. 6. Parallel inter-level repair: case $z \neq \perp \wedge m(z)|_\ell = m(x)|_\ell$.

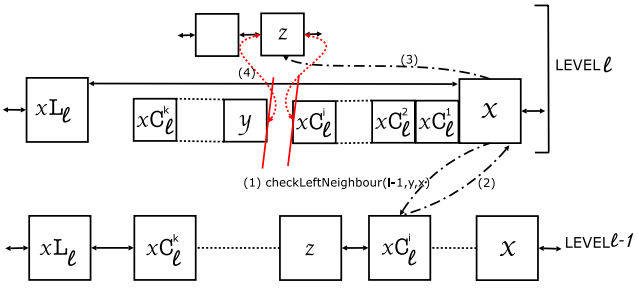

Fig. 7. Parallel inter-level repair: case $z \neq \perp \wedge m(z)|_\ell \neq m(x)|_\ell$.

4. Algorithms for range queries

Skip tree graphs and skip graphs preserve the order of the keys identifying each peer and then, support order-based queries. It is assumed that each node v stores a set of data objects identified by keys, which belong to the interval $[k_{vL_0}, k_v]$ where k_{vL_0} is the key of the left neighbour at bottom level and k_v is the key identifying the node.

In the cases where the query involves finding a unique result, the operation can be performed as an adaptation of the exact-match search operation [17] and while the performance is of the same order in both structures ($O(\log n)$ time with $O(\log n)$ messages), it was shown that skip tree graph cost is lower.

In this paper, the description and analysis of range queries requiring all keys of an interval for skip graphs and skip tree graphs are presented. The range query cost (in terms of the number of messages and hops) depends on the total number of peers in the network (n) and the number of peers intersected by the query (r). The latter amount is dependent on the length of the interval $|[\alpha, \beta]|$ defined by the range query. Clearly, $r \leqslant n$.

Three different range query schemes are presented for skip graphs: *sequential scheme*, *memory-less broadcasting scheme* [17], and *broadcasting with memory scheme*. All these schemes can be applied over a skip tree graph, as it is a super set of a skip graph. A specific scheme is designed for skip tree graphs, which exploits the underlying tree structures. Analytical results are presented justifying the costs of these schemes. It results that skip tree graph range query scheme outperforms all the rest. The next section presents experimental results that validate the theoretical results.

Fig. 8 shows an example how each of the schemes works over a subset of nodes from the skip tree graph of Fig. 2. The starting node is 20 and the range is [40,70]. Different arrows are used for each scheme and those corresponding to a message are labeled with the corresponding hop number.

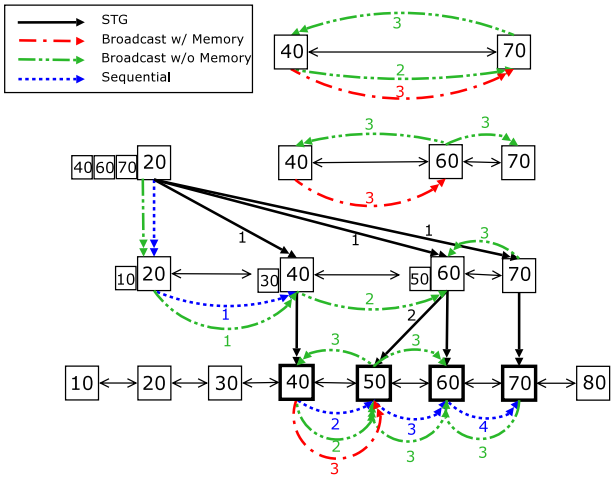

Fig. 8. Range-query schemes for Skip graph and Skip Tree Graph for the range query starting at node 20 for the range [40, 70]. Only a subset of nodes related with the range query are shown. Different arrows are used for each scheme, which are labeled with the corresponding hop number.

4.1. Skip graph schemes

The basic range query scheme that can be implemented over skip graphs involves finding the peer holding the lower bound (an operation which takes expected $O(\log n)$ hops and messages) and then, sequentially traversing the bottom level until the peer holding the upper bound is found. This scheme has been the only one considered in some works on range queries when comparing with the skip graph structure [28,29]. The sequential search has a cost of $O(r)$ messages and hops, where r is the number of nodes in the interval. Thus, the sequential approach takes expected $O(\log n + r)$ messages and hops.

Aspnes and Shah [17] envisioned a broadcasting approach for finding all the keys in a given interval when introducing the skip graph structure, whose cost was expressed as expected $O(r \log n)$ messages and $O(\log r)$ hops or time. This scheme amounts to reaching a node holding a single element in the interval, for instance the lower bound, and then at each following step, each node broadcasts the range query to all the nodes it knows belonging to the range. The node holding the lower bound forwards the query to all the nodes in its routing table belonging to the range (bounded by expected $O(\log r)$ messages and hops). Each node receiving the range query repeats the same process (keeping track of the query messages already received so that only one result is sent and unnecessary result messages are avoided). Then, the bound on the expected number of messages is $O(r \log r)$ ($O(\log r)$ for each of the r nodes). As regards the number of hops, traversing the range requires at most $O(\log r)$ hops. This bound can be obtained by considering that in the (sub) skip graph containing only the r peers in the range, any node can be reached in $O(\log r)$ hops [17]. To sum up, the range search in skip graph will take $O(\log n + r \log r)$ messages and $O(\log n + \log r)$ time. Note that given that $r \leqslant n$, $O(\log n + \log r)$ is $O(\log n)$. In addition, $O(\log n + \log r)$ is equivalent to $O(\log rn)$ (i.e., $O(\log n + \log r) = O(\log rn) = O(\log n)$). Although this scheme improves the sequential scheme with respect to latency, the number of messages required increases significantly.

The previous scheme, introduced by Aspnes and Shah [17], is denominated memory-less broadcasting (or equivalently, broadcasting without memory) in this paper, as opposed to broadcasting with memory which is described next. The improvement consists in observing that peers in the range may be receiving the range query several times, and the number of messages can be reduced by keeping track of the query messages already sent to other peers. Then, each message stores the list of nodes that have been visited and that will be visited in the next step. Therefore, when each node sends a query range, it will include all the neighbours in the range to which it is broadcasting the message and in this way, subsequent nodes will not

resend it to nodes already in the path. This scheme improves broadcasting without memory while it has the same asymptotic behaviour, requiring an expected number of $O(\log n + r\log r)$ messages and $O(\log n + \log r)$ hops. The experimental results show that these bounds have a smaller constant of proportionality than in the case of memory-less broadcasting. It is noted that communication overhead is not being considered but only number of messages.

4.2. Skip tree graph scheme

The implementation of range searches in skip tree graphs harnesses the parallelism inherent in P2P by relying on the underlying tree structures. The pseudo-code for the range search operation over skip tree graphs is given in Algorithm 1. The range query procedure can start on any node of the network. A node receiving a `rangeSearchOp` for a particular range $[\alpha,\beta]$ at level ℓ computes the interval of the identifier space for which it is responsible at that level (procedure `getOwnRange`(*level*)). The node's interval lower bound at level ℓ is given by the key of the rightmost conjugate if it exists, or the left neighbour's key otherwise, i.e. if it has no conjugate nodes at that level (in which case, the rightmost conjugate will be $\perp$). If this node is in the range and the procedure is at the bottom level, the result is sent to the starting node. If the operation is at level $\ell \geqslant 1$, the node establishes its ℓ-conjugate nodes that intersect the range $[\alpha,\beta]$ by means of the local procedure `getConjugatesToPropagateRange`(*range*, *level*). It then multicasts the range query operation to all these conjugates, indicating the immediate lower level. If the node itself is in the range, it will also continue the operation at its own immediate lower level.

The range search operation starts at the highest level of a node and in each step, the node will send messages at most to all its conjugate nodes, in the immediate lower level. As in each step the messages go down a level, the number of hops is directly related to the height of the skip tree graph, which is bounded by $O(\log n)$ on average. Theorem 3 states more precise bounds for the cost of the range search showing their dependency with the parameters n and r.

Theorem 3. *The range search operation in a skip tree graph STG with n nodes takes expected* $O(\log n + r)$ *messages and* $O(\log n + \log r) = O(\log n)$ *hops.*

At each level of the range query operation for the interval $[\alpha,\beta]$, which intersects r nodes, a node sends messages to all its conjugate nodes that intersect the range and this step is counted as one hop. Then, the number of messages and number of hops in a range query for the interval are related with the number of conjugate nodes accessed by the range search operation at each level $\ell \geqslant 1$. Let C_ℓ be a random variable representing such quantity.

In order to present the proof for Theorem 3, some intermediate results are presented in the following Lemmas. Lemma 4 enunciates the expression for the expected value of the random variables in the sequence $\{C_\ell\}_{\ell\geqslant 1}$.

Lemma 4. *The expected value of* C_ℓ *is* $E[C_\ell] = (1-p)[p^{\ell-1}(r-1)+1]$, *for all* $\ell \geqslant 1$.

Algorithm 1. Range search operation in node v over Skip Tree Graphs

```
1: upon receiving ⟨ rangeSearchOp, startNode, range, level ⟩:
2: while (level ⩾ 0) do
3:    ownRange ← getOwnRange(level)
4:    overlaps ← ownRange.overlaps(range)
5:    if (level = 0) then
6:       if overlaps then
7:          send⟨ foundElementOp, v ⟩ to startNode
8:       end if
9:       return
10:   end if
11:   conjugates ← getConjugatesToPropagateRange(range, level)
12:   send_parallel⟨ rangeSearchOp, startNode, range, level-1 ⟩ to conjugates
13:   if overlaps then
14:      level ← level − 1
15:   else
16:      return
17:   end if
18: end while
```

```
getOwnRange (level): KeyRange
 rightmostConjugate ← getRightmostConjugate()
 leftNeighbour ← neighbour[LEFT][level]
 if rightmostConjugate ≠ ⊥ then
    return (rightmostConjugate.key, v.key]
 end if
 if leftNeighbour ≠ ⊥ then
    return (leftNeighbour.key, v.key]
 end if
 return (v.key, v.key]
```

Lemmas 5 and 6 establish upper bounds for the expected values of summations related to the sequence of random variables $\{C_\ell\}_{\ell\geqslant 0}$.

Lemma 5. $\sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} E[C_\ell] = O(\log_{1/p} n + r)$ *and* $E[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_i] = O(r)$, *where H is the random variable representing the height of the skip tree graph.*

Lemma 6. *Let be L a random variable denoting the number of levels for which the range query does not access any conjugate node* (*i.e. it counts the levels* ℓ *such that* $C_\ell = 0$). *Then,* $E[L] = O(\log_{1/p} n + \log_{1/p} r)$.

The proofs for all the lemmas are presented in Appendix A.

Proof of Theorem 3. Let M be the random variable denoting the number of messages sent by the range query. Thus, $M = \sum_{\ell=1}^{H} C_\ell$. Since the number of terms in this summation is the random variable H for the height of the skip tree graph, linearisation properties of the expected value cannot be applied ($E[M]$ is not equal to $\sum_{i=1}^{H} E[C_i]$). Additionally, the sequence of random variables H, C_1, $C_2, \ldots$ is not mutually independent and not even pairwise independent. Then, in order to bound the expected value of the number of messages, the value of M is expressed as follows:

$$M = \sum_{i=1}^{\lceil \log_{1/p} n \rceil} C_i + \sum_{i=\lceil \log n \rceil + 1}^{H} C_i \tag{1}$$

As the first summation has a fixed number of terms, it follows that

$$E[M] = \sum_{\ell=1}^{\lceil \log_{1/p} n \rceil} E[C_\ell] + E\left[\sum_{\ell=\lceil \log n \rceil + 1}^{H} C_i \right] \tag{2}$$

Each of these summations is bounded by Lemma 5 and it results that the expected number of messages in the range search is $O(\log_{1/p} n + r)$.

As regards the number of hops, it is clear that they are directly related with the height of the skip tree $O(\log n)$, as in each step the range search operation goes down a level.

To obtain a more precise bound, T is defined as the random variable for the number of hops required for the range query. Then, the value of T is determined by considering all the levels in the skip tree graph and subtracting the levels for which no conjugate nodes are accessed by the range query. In symbols, $T = H - L$, where L and its expectation are as defined in Lemma 6.

Then,

$$E[T] = E[H] - E[L] = O(\log_{1/p} n) - O(\log_{1/p} n - \log_{1/p} r) \tag{3}$$

$$= O(\log_{1/p} n + \log_{1/p} r) = O(\log n). \quad \square \tag{4}$$

4.3. Discussion

Table 1 displays the costs of each of the schemes presented in relation to the average number of messages and average number of hops required. Broadcasting schemes, with or without memory, improve with respect to the sequential scheme in the number of hops. However, the number of messages that need to be sent increases. Although both broadcasting schemes have the same asymptotic performance, the experimental results in Section 4.4 show that the constant of proportionality for broadcasting with memory is less than the corresponding one for broadcasting without memory. Hence, the cost in number of messages and hops is ameliorated by keeping track of the nodes visited. The range query scheme implemented for skip tree graph reduces the number of messages so that its asymptotic behaviour matches that of the sequential scheme. The simulations' results presented in the next section show that the constant of proportionality is slightly smaller for the parallel scheme than the sequential scheme in the number of messages. As a result, the range query scheme for skip tree graph outperforms the rest of the schemes with respect to both performance costs.

Table 1
Asymptotic behaviour of the range query schemes with respect to average number of messages and hops

Range query scheme	DST	Messages	Hops
Sequential	SG	$O(\log n + r)$	$O(\log n + r)$
Memory-less broadcasting	SG	$O(\log n + r \log r)$	$O(\log rn) = O(\log n)$
Broadcasting with memory	SG	$O(\log n + r \log r)$	$O(\log rn) = O(\log n)$
Tree-based	STG	$O(\log n + r)$	$O(\log_r n) = O(\log n)$

As regards concurrency issues related to guaranteeing the correctness of the range-query results, Linga et al. [28] have developed techniques to guarantee accuracy and availability of P2P range indices. Their work only considers that the range search is performed sequentially, by finding the lower bound and then scanning along the bottom list to retrieve the items in the range [28], with skip graph being one of their examples. Their analysis focuses on maintaining consistency of the bottom level and adapting to changes in the ranges maintained by the peers. In the case of the schemes discussed here, it is observed that broadcasting approaches may be accessing all the levels of the skip graph at the same time. Then, correct results will be obtained if all the rings are consistent, heavily limiting concurrency and scalability. On the other hand, the skip tree graph scheme traverses each level at a time. This fact implies that inconsistencies can be treated separately at each level and the degree of concurrency achieved will be higher.

4.4. Experimental evaluation

The skip tree graph data structure was implemented and evaluated under event-driven simulations. The experiments were performed to validate the analytical results related to the different range query schemes presented in the previous sections. Three sets of experiments were undertaken to show how the performance cost, in terms of the average number of messages and average number of hops required, is influenced by each of the parameters involved in a range query $[\alpha, \beta]$: the number of nodes in the network (n), the number of nodes intersecting the range (r, which are the nodes storing the result set) and the length of the range $|[\alpha, \beta]|$. The results are shown in the three following subsections.

The experimental methodology consisted in building 1000 skip tree graphs for each set of parameters and measuring in each of them the cost of performing range queries for each scheme. The starting nodes for the range queries were chosen at random. For all the simulations,

the identifier space was considered as the interval $K = [0, 10000]$. The peers' identifiers were distributed uniformly at random in the identifier space. Regression analysis was used to determine the relationship between the parameters in the experiments to show that the results are in agreement with the theoretical results.

In all the plots, the curve for the sequential scheme is exactly the same when considering number of messages and number of hops, as its cost is equivalent in both measures. However, as different scales were used for the corresponding graphics, the reader needs to take this into account. The reason why different scales were used is because the focus is on comparing each scheme with respect to either the number of messages or the number of hops.

4.4.1. Network size fixed

For a network size fixed at 1000 nodes, experiments were run for range lengths varied from 20 to 500 in steps of 20. The cost of the different range query schemes was measured in terms of messages (Fig. 9a) and hops (Fig. 9b).

Since the identifiers for the 1000 nodes are chosen uniformly at random in the same identifier space, it results that r varies linearly with respect to the range length $|[\alpha, \beta]|$ ($r \sim |[\alpha, \beta]|n/|K|$ and the greater the range length, the greater the number of nodes intersecting the interval $[\alpha, \beta]$). Consequently, the functions in the plots can be seen as functions of r. In the case of the number of messages, regression verified that the sequential and tree-based range query schemes are linear with respect to r and both broadcasting schemes follow the function $r \log r$. For the number of hops, all the schemes have a logarithmic cost with respect to r, except from the sequential scheme that is linear. The skip tree graph range query scheme outperforms the rest in both measurements.

4.4.2. Range size fixed (on average)

The network size n was varied in the interval $[10, 1000]$ in steps of 50. The range length was chosen for each n so that $r \sim 10$ nodes on average (given the uniform distribution of nodes in K). The plots in Fig. 10a and b show that the number of messages and hops depend logarithmically on n for all the range query schemes. The skip tree graph scheme outperforms the rest in both cases.

4.4.3. Range length fixed

In the last set of experiments, the network size was varied from 50 to 1000 in steps of 50. In order to observe the behaviour of the cost functions near the origin, an additional point was considered for $n = 10$ nodes. The range length $|[\alpha, \beta]|$ was fixed at 500. Given that peers' identifiers are randomly and uniformly distributed in the identifier space, as n increases, the number of peers intersecting the range increases. Therefore, it results that there is a linear relation between n and r. Therefore, the simulations' results show the costs as a function of x, with n and r linearly dependent on x. Regression analysis showed that the functions in Fig. 11a and b are in agreement with the theoretical results summarised in Table 1. The number of messages and hops for the sequential scheme follow exactly the same function $\log x + x$. The number of messages for the skip tree graph scheme also follows this function $\log x + x$ with a slightly smaller constant of proportionality. Both broadcasting approaches require a number of messages which varies as $\log x + x \log x$. On the other hand, the number of hops for skip tree graph and both broadcasting schemes follows a logarithmic function. Range queries over skip tree graph show the best cost in terms of messages and hops with n and r variable.

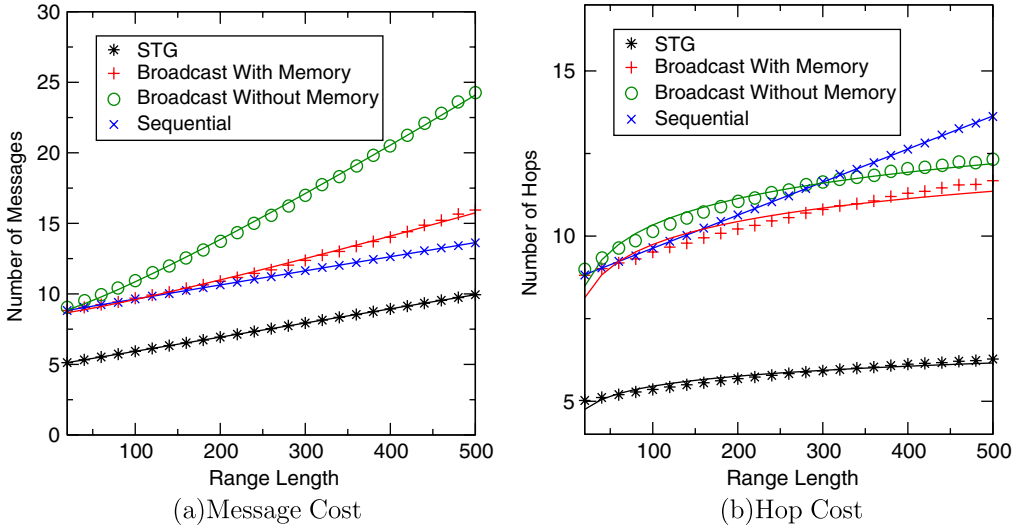

Fig. 9. Cost in messages and hops for network size fixed, with r variable.

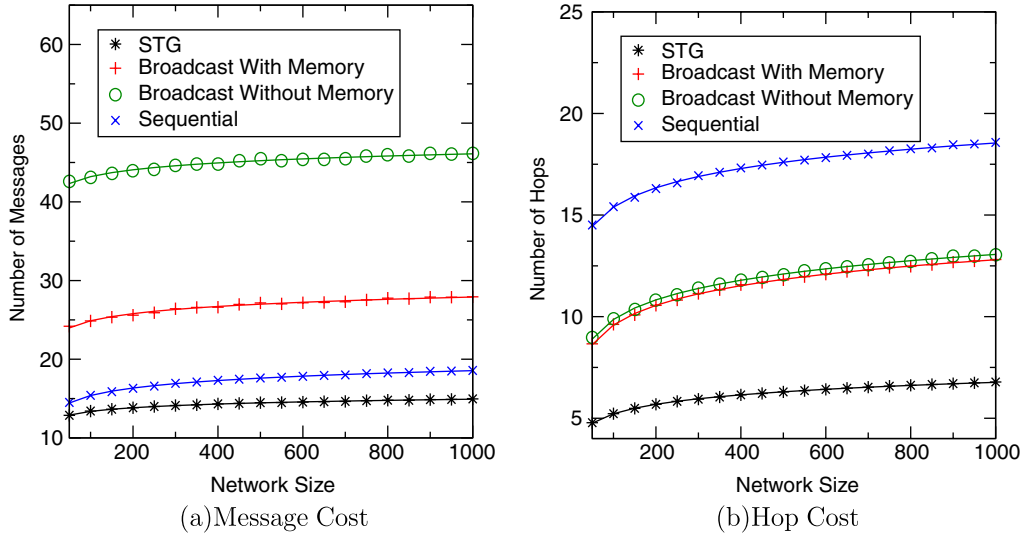

Fig. 10. Cost in messages and hops for range size fixed.

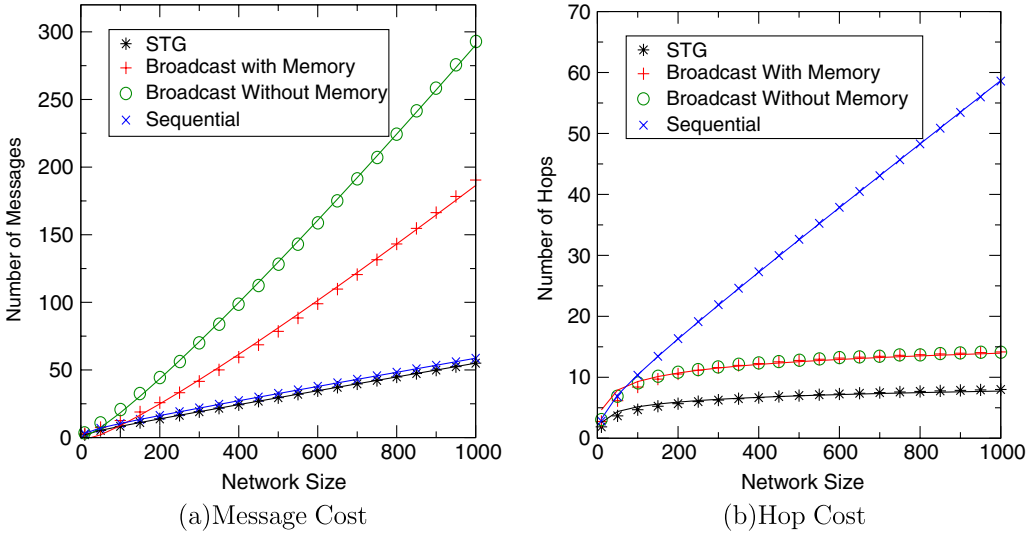

Fig. 11. Cost in messages and hops for range length fixed.

5. Conclusions

This paper contributes to the currently active and important area of research dealing with the provision of distributed data structures supporting efficient complex queries over structured peer-to-peer networks, including range, prefix and aggregation queries. The skip tree graph structure has been formally presented as an isomorphic extension of skip graphs. While skip graphs are based on skip lists, skip tree graphs are based on skip trees, a concurrent approach to skip lists. Skip tree graph's properties and algorithms have been thoroughly analysed and compared to its predecessor. While skip graphs support order-based queries such as range and prefix queries, they do not support aggregation queries, functionality which is provided by skip tree graphs. This extension of the functionality is allowed by the underlying hierarchical structure of skip tree graphs, which additionally permits of a significant improvement on the repair mechanism and straightforward implementation of multicast/broadcast operations. It has also been shown that efficiency on exact-match searches is significantly improved. Furthermore, existing range-query schemes for skip graphs were compared with new proposed schemes over both structures. Analytical results as well as experimental evaluation via discrete-event simulations have demonstrated that skip tree graph performs exact-match and range queries efficiently and they outperform the respective operations in skip graphs.

Acknowledgments

The authors would like to thank the Academic Planning Group at Queen's University Belfast for funding the work

of Alejandra González-Beltrán and the anonymous reviewers for their useful comments.

Appendix A

Proof of Lemma 1. In the insertion procedure, when determining the connections for node u at level ℓ in the right direction, $|\Sigma| = p^{-1}$ nodes $\{v_\sigma\}_{\sigma\in\Sigma}$ are searched, one for each of the symbols in the alphabet Σ such that $m(v_\sigma)_{\ell+1} = \sigma$. Each v_σ may or may not exist. In the latter case, the whole list at level ℓ will be traversed. The node w such that $m(w)_{\ell+1} = m(u)_{\ell+1}$ is u's right neighbour at level $\ell+1$ and the rest are used to add u as conjugate in $p^{-1}-1$ lists.

Let X_σ be a random variable counting the number of steps required to find a node v_σ such that $m(v)_{\ell+1} = \sigma$, for $\sigma \in \Sigma$. Then, the number of messages and hops needed for u to find the p^{-1} nodes is denoted by the variable $X = \max_{\sigma\in\Sigma} X_\sigma$ where $\{X_\sigma\}$ are not independent. However, as they are geometric random variables with $E[X_\sigma] = p^{-1}$, an upper bound on $E[X]$ is obtained as $|\Sigma|p^{-1} = p^{-2}$. □

Proof of Lemma 4. Each random variable C_ℓ, defined as the number of conjugates accessed by the range search operation at level $\ell \geqslant 1$, is dependent on the number of nodes accessed by the range query in the immediate lower level. Let N_ℓ be a random variable denoting the number of nodes accessed by the range query at level ℓ.

In order to determine M's value, the two sequences of random variables $\{N_\ell\}_{\ell\geqslant 0}$ and $\{C_\ell\}_{\ell\geqslant 0}$ must be analysed. With that objective, the levels of the STG are considered in a bottom-up fashion. In the bottom level, the range query accesses r nodes (i.e., $N_0 = r$) and no conjugate nodes ($C_0 = 0$).

At the first level, N_1 and C_1 are the cardinality of two random subsets of N_0: R_1 and $\overline{R_1}$. The intuition is that as there is a probability p that each of the nodes appears in the next level as a node and $1-p$ as a conjugate node. Then, the expected number of C_1 is $(1-p)r$. In the case of N_1, it may happen that the rightmost node, say v_r, among the r nodes at the bottom level is a conjugate node. In this case, an extra node in the first level (not included in the r nodes in the result) is accessed by the range search operation. Consequently, one node is added to N_1 regardless of v_r appearing as a node in level 1 or not. The expected number of N_1 is $p(r-1)+1$.

The preceding reasoning is formalised by considering an indicator random variable I_i for each i, $1 \leqslant i \leqslant r$, indicating if v_i appears as a node at level 1 or not:

$$I_i = \begin{cases} 1, & v_i \in \{\text{nodes accessed by the range query in level 1}\} \\ 0, & \text{otherwise} \end{cases}$$

whose expected value is $E[I_i] = 0Pr[I_i = 0] + 1\ Pr[I_i = 1] = Pr[I_i = 1] = p$.

Then, the size N_1 of the random subset R_1 is equal to $N_1 = \sum_{i=1}^{r-1} I_i + 1$.

This results from considering that I_i counts if each of the first $r-1$ nodes appears as a node at level 1 and summing one unit regardless of the rightmost node appearing as a node or a conjugate (because an extra node will be used in that case).

Consequently, the expected value is $E[N_1] = E[\sum_{i=1}^{r-1} I_i + 1] = \sum_{i=1}^{r-1} E[I_i] + 1 = \sum_{i=1}^{r-1} p + 1 = (r-1)p + 1$.

In the case of the number of conjugate nodes, it follows that $C_1 = \sum_{i=1}^{r}(1-I_i)$ and then the expected value is $E[C_1] = E[\sum_{i=1}^{r}(1-I_i)] = r(1-p)$.

The subsequent values for N_ℓ and C_ℓ, with $\ell \geqslant 2$, depend on $N_{\ell-1}$. It is important to note that N_ℓ and C_ℓ represent the size of a random subset of a set that is also random. Thus, the same reasoning as before cannot be applied. First, N_2 and C_2 values are considered.

Using the conditional expectation $E[X] = \sum_y E[X|Y = y]Pr[Y = y]$, it results

$$\begin{aligned} E[N_2] &= \sum_{k=0}^{r} E[N_2|N_1 = k]Pr[N_1 = k] \\ &= \sum_{k=0}^{r} [p(k-1)+1]Pr[N_1 = k] \\ &= p\sum_{k=0}^{r} kPr[N_1 = k] + (1-p)\sum_{k=0}^{r} Pr[N_1 = k] \\ &= pE[N_1] + (1-p)1 \\ &= p^2(r-1) + 1 \end{aligned}$$

$$\begin{aligned} E[C_2] &= \sum_{k=0}^{r} E[C_2|N_1 = k]Pr[N_1 = k] \\ &= \sum_{k=0}^{r} k(1-p)Pr[N_1 = k] \\ &= (1-p)E[N_1] = (1-p)[(r-1)p+1] \end{aligned}$$

Following the same reasoning, and by induction, it follows that the expected number of nodes accessed by the range search operation at level ℓ is

$$E[N_\ell] = p(E[N_{\ell-1}] - 1) + 1 \tag{A.1}$$

Thus, it can be proved by induction that

$$E[N_0] = r \tag{A.2}$$

$$E[N_\ell] = p(E[N_{\ell-1}] - 1) + 1 \tag{A.3}$$

implies $E[N_\ell] = p^\ell(r-1) + 1$.

The expected number of conjugates accessed by the range search operation at level ℓ is such that

$$E[C_\ell] = (1-p)E[N_{\ell-1}] \tag{A.4}$$

Therefore, $E[C_\ell] = (1-p)[p^{\ell-1}(r-1)+1]$. □

Proof of Lemma 5. First, an upper bound on the first summation is found:

$$\sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} E[C_\ell] = \sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} [(1-p)(p^{\ell-1}(r-1)+1)] \quad (A.5)$$

$$= \sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} [(1-p)p^{\ell-1}(r-1)+(1-p)] \quad (A.6)$$

$$= \frac{(1-p)(r-1)}{p} \sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} p^{\ell} + (1-p) \sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} 1 \quad (A.7)$$

Recalling the formula for the sum of a geometric sequence $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}$ and for the sum index starting in 1, $\sum_{i=1}^{n} c^i = \frac{c(c^n-1)}{c-1}$, it results

$$\sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} E[C_\ell] = \frac{(1-p)(r-1)}{p} \frac{p(p^{\lceil \log_{1/p} n\rceil}-1)}{(p-1)} + \lceil \log_{1/p} n\rceil (1-p) \quad (A.8)$$

$$= (r-1)(1-p^{\lceil \log_{1/p} n\rceil}) + \lceil \log_{1/p} n\rceil (1-p) \quad (A.9)$$

And as $p^{\lceil \log_{1/p} n\rceil} \geqslant p^{\log_{1/p} n} = \frac{1}{n}$, it results that $1 - p^{\lceil \log_{1/p} n\rceil} \leqslant 1 - \frac{1}{n}$

$$\sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} E[C_\ell] \leqslant (r-1)\left(1-\frac{1}{n}\right) + \lceil \log_{1/p} n\rceil (1-p) \quad (A.10)$$

$$= (1-p)\lceil \log_{1/p} n\rceil + (r-1)\left(1-\frac{1}{n}\right) \quad (A.11)$$

Considering

$$\lceil \log_{1/p} n\rceil \leqslant \lfloor \log_{1/p} n\rfloor + 1 \leqslant \log_{1/p} n + 1 \quad (A.12)$$

it follows that $\lceil \log_{1/p} n\rceil = O(\log_{1/p} n)$.

Additionally, as $\lim_{n\to\infty} \frac{1}{n} = 0$, it results $(r-1)(1-\frac{1}{n}) = O(r)$.

The conclusion is that the first summation $\sum_{\ell=1}^{\lceil \log_{1/p} n\rceil} E[C_\ell] = O(\log_{1/p} n + r)$.

Next, a bound on the second summation $E[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_i]$ is found.

The summation is non-zero if and only if $H \geqslant \lceil \log n\rceil + 1$. Since H is random the lemma on the conditional expected value of the summation given the value of H is used.

Let I_H be a random variable whose value is equal to

$$I_H = \begin{cases} 1, & \text{if} H \geqslant \lceil \log n\rceil + 1 \\ 1, & \text{if} H \leqslant \lceil \log n\rceil \end{cases}$$

Then,

$$E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell\right] = E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell | I_H = 1\right] Pr[I_H = 1] \quad (A.13)$$

$$+ E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell | I_H = 0\right] Pr[I_H = 0] \quad (A.14)$$

If $I_H = 0$, i.e. $H \leqslant \lceil \log n\rceil$, the summation is zero. Moreover, $Pr[I_H = 1] \leqslant 1$. Incorporating these facts into the equation

$$E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell\right] \leqslant E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell | I_H = 1\right] \quad (A.15)$$

And considering that for any ℓ, $C_\ell \leqslant r$, it follows that

$$E\left[\sum_{\ell=\lceil \log n\rceil+1}^{H} C_\ell | I_H = 1\right] \leqslant E\left[(H - \lceil \log_{1/p} n\rceil) r\right] \quad (A.16)$$

$$= r(E[H] - \lceil \log_{1/p} n\rceil) \quad (A.17)$$

$$\leqslant r(\lceil \log_{1/p} n\rceil + 1 - \lceil \log_{1/p} n\rceil) \quad (A.18)$$

$$\leqslant r \quad \square \quad (A.19)$$

Proof of Lemma 6. Let L_ℓ, $\ell \geqslant 1$ be an indicator random variable such that

$$L_\ell = \begin{cases} 0, & C_\ell \geqslant 1 \\ 1, & C_\ell < 1 \end{cases}$$

whose expected value is

$$E[L_\ell] = 0 Pr[C_\ell \geqslant 1] + 1 Pr[C_\ell < 1] = Pr[C_\ell < 1] \quad (A.20)$$

Then, $L = \sum_{\ell=1}^{H} L_\ell$ and its expected value is $E[L] = \sum_{\ell=1}^{\lceil \log_{1/p} r\rceil} E[L_\ell] + E[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell]$. The next step is to bound each of the summations.

Using the expected value of L_ℓ from Eq. (A.20) and the fact that $Pr[C_\ell < 1] \leqslant 1$, it follows that:

$$\sum_{\ell=1}^{\lceil \log_{1/p} r\rceil} E[L_\ell] = \sum_{\ell=1}^{\lceil \log_{1/p} r\rceil} Pr[C_\ell < 1] \leqslant \sum_{\ell=1}^{\lceil \log_{1/p} r\rceil} 1 = \lceil \log_{q/p} r\rceil \quad (A.21)$$

With respect to the second summation, the conditional expectation is applied using the indicator random variable I_H as in the proof of the number of messages.

$$E\left[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell\right] = E\left[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell | I_H = 1\right] Pr[I_H = 1] \quad (A.22)$$

$$+ E\left[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell | I_H = 1\right] Pr[I_H = 0] \quad (A.23)$$

$$E\left[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell\right] \leqslant E\left[\sum_{\ell=\lceil \log_{1/p} r\rceil+1}^{H} L_\ell | I_H = 1\right] \quad (A.24)$$

$$\leqslant E[H - \lceil \log_{1/p} r\rceil] = E[H] - E[\lceil \log_{1/p} r\rceil] \quad (A.25)$$

$$\leqslant \lceil \log_{1/p} n\rceil + \frac{1}{1-p} + \lceil \log_{1/p} r\rceil \quad (A.26)$$

This means that the second summation is $O(\log_{1/p} n + \log_{1/p} r)$ and $E[L] = O(\log_{1/p} n + \log_{1/p} r)$. $\square$

References

[1] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, San Diego, CA, USA, 2001, pp. 149–160.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, A scalable content-addressable network, in: Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies,

Architectures, and Protocols for Computer Communication, ACM Press, San Diego, CA, USA, 2001, pp. 161–172.
[3] A. Rowstron, P. Druschel, Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001, pp. 329–350.
[4] B.Y. Zhao, L. Huang, J. Stribling, S. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: A Resilient Global-scale Overlay for Service Deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 41–53.
[5] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, in: Proceedings of the Twenty-Ninth ACM Symposium on Theory of Computing (STOC), El Paso, TX, USA, 1997, pp. 654–663.
[6] M. Harren, J. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, I. Stoica, Complex Queries in DHT-based Peer-to-Peer Networks, in: Proceedings of the First International Workshop on P2P Systems (IPTPS'02), Springer, 2002, pp. 242–259.
[7] N. Daswani, H. Garcia-Molina, B. Yang, Open problems in data-sharing peer-to-peer systems, in: Proceedings of the Ninth International Conference on Database Theory, Springer, 2003, pp. 1–15.
[8] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
[9] A. Andrzejak, Z. Xu, Scalable, efficient range queries for grid information services, in: Proceedings of IEEE International Conference on Peer-to-Peer Computing, P2P 2002, pp. 33–40.
[10] Y. Shu, B.C. Ooi, K.-L. Tan, A.Y. Zhou, Supporting multi-dimensional range queries in peer-to-peer systems, in: Proceedings of the Fifth International Conference on Peer-to-Peer Computing, 2005, pp. 173–180.
[11] I. Abraham, J. Aspnes, J. Yuan, Skip B-Trees, in: Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS), Pisa, Italy, 2005, pp. 284–295.
[12] A. Gupta, D. Agrawal, A.E. Abbadi, Approximate range selection queries in peer-to-peer systems, in: Proceedings of the First Biennal Conference on Innovative Data Systems Research (CIDR), 2003, pp. 141–151.
[13] T. Pitoura, N. Ntarmos, P. Triantafillou, Replication, load balancing and efficient range query processing in DHTs, in: Proceedings of the 10th International Conference on Extending Database Technology (EDBT), LNCS, 2006, pp. 131–148.
[14] A. Crainiceanu, P. Linga, J. Gehrke, J. Shanmugasundaram, Querying peer-to-peer networks using P-Trees, in: Proceedings of the Seventh International Workshop on the Web and Databases (WebDB 2004), Paris, France, 2004, pp. 25–30.
[15] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, J. Hellerstein, A case study in building layered DHT applications, in: Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2005, pp. 97–108.
[16] K. Aberer, P-Grid: a self-organizing access structure for p2p information systems, in: Proceedings of the 6th Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy, 2001, pp. 179–194.
[17] J. Aspnes, G. Shah, Skip Graphs, in: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 2003, pp. 384–393.
[18] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, A. Wolman, SkipNet: A Scalable Overlay Network with Practical Locality Properties, in: Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS'03), Seattle, WA, USA, 2003, pp. 113–126.
[19] H.V. Jagadish, B.C. Ooi, Q.H. Vu, BATON: A Balanced Tree Structure for Peer-to-Peer Networks, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway, 2005, pp. 661–672.
[20] H.V. Jagadish, B.C. Ooig, K.-L. Tan, Q.H. Vu, R. Zhang, Speeding up search in peer-to-peer networks with a multi-way tree structure, in: Proceedings of SIGMOD 2006, Chicago, Illinois, USA, 2006, pp. 1–12.
[21] W. Pugh, Skip Lists: A probabilistic alternative to balanced trees, Communications of the ACM 33 (6) (1990) 668–676.
[22] X. Messeguer, Skip Trees: an alternative data structure to Skip Lists in a concurrent approach, Informatique Théorique et Applications 31 (3) (1997) 251–269.
[23] A. González-Beltrán, P. Sage, P. Milligan, Skip Tree Graph: a distributed and balanced search tree for peer-to-peer networks, in: Proceedings of the IEEE International Conference on Communications, Glasgow, UK, 2007.
[24] O.D. Sahin, A. Gupta, D. Agrawal, A.E. Abbadi, A peer-to-peer framework for caching range queries, in: International Conference on Data Engineering (ICDE), 2004, pp. 165–176.
[25] A. Datta, M. Hauswirth, R. John, R. Schmidt, K. Aberer, Range queries in trie-structured overlays, in: Proceedings of the Fifth International Conference on Peer-to-Peer Computing, 2005, pp. 57–66.
[26] J. Aspnes, J. Kirsch, A. Krishnamurthy, Load balancing and locality in range-queriable data structures, in: Proceedings of the 23rd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2004), St. John's, Newfoundland, Canada, 2004, pp. 115–124.
[27] T. Papadakis, Skip lists and probabilistic analysis of algorithms, Ph.D. thesis, University of Waterloo, 1993.
[28] P. Linga, A. Crainiceanu, J. Gehrke, J. Shanmugasundaram, Guaranteeing correctness and availabitlity in P2P range indices, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, 2005, pp. 323–334.
[29] D. Li, J. Cao, X. Lu, K.C.C. Chan, B. Wang, J. Su, H. va Leong, A.T.S. Chan, Delay-bounded range queries in DHT-based peer-to-peer systems, in: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), 2006, p. 64.

Alejandra González-Beltrán received the degree of Licentiate in Computer Science (Honours) from Universidad Nacional de Rosario, Argentina. She is currently working towards a Ph.D. in Computer Science at the School of Electronics, Electrical Engineering and Computer Science at Queen's University Belfast, Northern Ireland, United Kingdom. Her research interests include distributed systems and software engineering. In particular, her current work is in the area of peer-to-peer networks and grid computing.

Dr Peter Milligan received the degrees of BSc Computer Science (Honours) and Ph.D. from the Queen's University of Belfast. Dr Milligan is a member of the School of Electronics, Electrical Engineering and Computer Science where he holds a senior lectureship and is chair of the School's Postgraduate Research Committee. Dr Milligan has worked in the field of High Performance and Distributed Computing for over 25 years with special interest in the development of environments for the automatic generation of code for parallel architectures. Current interest is focused on grid and peer-to-peer systems with the goal of developing middleware that will assist with, e.g., resource discovery, dynamic task/resource mapping, information retrieval in small worlds, efficient management of large data sets, and the relationship between regulation and trust in virtual organisations. Dr Milligan is on the editorial boards of three journals and has been a member of over 40 international programme, scientific and conference committees.

Dr Paul Sage received the degrees of B.Sc. Computer Science (Honours) and Ph.D. from the Queens University of Belfast and is a member of the School of Electronics, Electrical Engineering and Computer Science where he currently holds a lectureship. His research interests include parallel, distributed and grid computing and is specifically interested in fault tolerance in unstable computational environments, virtual execution environments and intelligent resource aggregation. Dr Sage has served on several international program committees and is the author of 50+ conference and journal publications in the field.