Search Algorithms for Distributed Data Structures in P2P Networks

Raed Al-Aaridhi, Iakov Dlikman, Newton Masinde, and Kalman Graffi
Department of Computer Science, University of Düsseldorf, Germany
Email: {alaaridhi, iakov.dlikman, newton.masinde, graffi}@hhu.de

***Abstract*—The search problem in distributed systems has been evaluated and various proposals have been made to solve them. The problem is particularly pronounced when considering P2P networks, because the different types of P2P network topologies used to dictate the search mechanisms that are used. Specifically, search mechanisms may be key-based, as in structured P2P networks that use distributed data structures such as Distributed Hash Tables (DHTs), Sets, Links, and Trees, or keyword-based for unstructured P2P networks. Online systems today utilize metadata to support search algorithms. Metadata searches are trivial in unstructured networks but are a challenge to the structured networks because of the distributed data structures (DDS). This work proposes to introduce algorithms that support metadata searches in structured P2P networks that utilize DHT overlays. DDSs are introduced into the system as storage layers that support complex data structures. We show that this can be done and searching can be achieved with acceptable performance.**

I. Introduction

Peer-to-peer (P2P) network technologies have been extensively researched in literature and have rapidly found usefulness in various interactive online applications, such as massive multiplayer games (MMGs) and online social networks (OSNs). However, from the onset, research reveals that P2P networks face some debilitating challenges that hamper the smooth functioning of the interactive online application. These challenges are the topology, storage, update handling, search and addressing, security and robustness to churn [1]. The discussion of all of the challenges is beyond the scope of this work. The aim of this research is to consider the problem of search and retrieval in the P2P network with an emphasis on metadata-based searches.

The search problem in P2P networks is the challenge of locating a needed resource within the network. A resource can be either a node or data item. In the P2P networks, unlike in centralized systems, the resource to be retrieved is not located at a central location. Therefore the searching algorithm must take into account the distributed nature and heterogeneity of nodes in the network. In an analysis of the distributed network environment in general, and P2P networks in particular, the search and retrieval of a resource is broken into three distinct stages: i) identification of needed resource based on metadata criteria (*Metadata Search*), ii) ascertaining the location of the data (*Document Location Search*), and iii) downloading the resource item (*Document Download*) [2], [3]. This problem can also be briefly looked at as involving resource description, resource selection and results fusion [4]. In essence, the lookup problem in distributed systems can be stated: "Given a data item X stored at some dynamic set of nodes in the system, find it." [5].

It has been highlighted in literature that solutions to be used to solve the p2p search and lookup problem must consider and address two important matters: *how p2p search networks work* and *how robust they are* [6]. At the heart of the solution to meet these concerns is developing of necessary indexing mechanisms (also referred to as query routing [7]) as well as querying mechanisms for efficient information retrieval. The resulting search techniques will then be dependable and adaptable to the changing network [6].

Indexing mechanisms on P2P systems are divided into *local*, *centralized* and *distributed* indexing [6]. Local indexes are indexes that are kept at the node for its own resources only. Centralized indexes depend on a server to maintain data references to the nodes that contain them. In this case, the index is centralized but data is distributed. For a true P2P network, the search mechanisms need to rely on distributed indexes. Distributed indexes rely on the use of pointers to forward the relevant resource queries to the target. The target pointers will be resident at several nodes. The distributed indexes are classified as a *semantic* and *semantic-free* mechanism.

Semantic mechanism is especially utilized in unstructured P2P networks. They are so called because they are human readable. One of their key advantages is that they highlight existing relationships thus work well with metadata in performing searches. However, they do not guarantee that scarce items will be found due to their reliance on heuristics. As no global index is constructed, semantic schemes cannot perform exact match searches, but keyword searches are possible. Therefore, it has been shown that the use of semantic routing allows each node in the network to specialize in a particular metadata area [3]. Semantic-free mechanisms rely heavily on the distributed data structures (DDSs), such as the distributed hash tables (DHTs), that are utilized to organize resources in the structured P2P networks. The use of the DHTs particularly, to provide semantic-free indexing is based on the performance characteristics of the DHTs. They provide efficient routing (typically $O(\log\ n)$, high scalability, self organization, incremental deployability (deployment without interrupting normal operations due to new joins) and are robust against node dynamics [8]. However, storing the metadata directly in the DHTs tends to bring about routing hotspots as nodes with popular metadata fields will have heavy routing traffic [3],

[8].

Considering the various methods of performing searches based on distributed index schemes, one challenge is to achieve keyword searches through metadata searches as well as exact match searches through utilization of key-based search mechanism. An example of a possible solution is found in the proposal by Dahan et al. [9], which introduced the Distributed Spanning Tree (DST) structure which performs a bottom-up approach in organizing peers in a tree topology. For meta data searching in new applications, Li et al. [10] introduced a cache sharing scheme for structured data for peers which are connect in mobile P2P networks, in which metadata of common data are distributed to the neighbors and next peers for efficient data searching. To perform data searches, queries are processed in the order of local cache, metadata, the cluster to which it belongs, and neighbor clusters, based on cooperative caching strategy. The authors in [11] propose an algorithm that reduces the search duration into milliseconds for a fingerprint (perceptual hash)-based distributed video identification system in peer-to-peer (P2P).

In previous work [12], [13], a DDS scheme based on DHTs was proposed, which consisted of linked data items. This DDS was basically pointer graphs in the DHT in which each linked data object contained ID information, pointers, and a payload. Further, the DDS supported key functions such as StoreDDS, RetrieveDDS, ModifyDDS (that is, Add Entry and Delete Entry). The proposed DDS scheme's evaluations had low overhead as well as low delay. It is noteworthy, that the functionality of this DDS will essentially be incomplete if an efficient mechanism for searching metadata is not considered. Therefore this work extends the DDS scheme proposed [12] by supporting metadata search algorithms to bolster the search functionalities.

The contributions of this paper are threefold:

- We introduce the concept and benefits of using Search mechanisms for finding metadata in DDSs in detail.
- We show through simulation the performance characteristics of search methods.
- We evaluate and compare different kinds of search mechanisms.

The remainder of the paper is organized as follows: Section II presents the search mechanisms for metadata in DDSs, highlighting their methods clearly. Section III gives an overview of the evaluation methods for the experimental work, and section IV gives and discusses the results. Finally, the conclusions are presented in section V.

II. Search Mechanisms for metadata in DDS in Overlay networks

This section gives a working definition of metadata and highlights key aspects of metadata research. Also, key techniques that were utilized in this research relating to how the searched metadata is aggregated into a single view via appropriate join techniques will be discussed. Three key techniques are reviewed that will be evaluated: LocalJoin, NetworkJoin and BloomJoin.

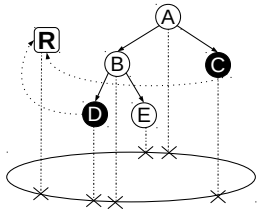


Fig. 1. DDS structure with corresponding DHT overlay

A. Defining Metadata

Metadata can generally be defined as data or information about file or document in the system. The presence of metadata supports six key functions in any data repository: i) creation, multiversioning, reuse, and recontextualization of information objects, ii) organization and description, iii) validation of the resources, iv) searching and retrieval, v) utilization and preservation, and finally, vi) documenting the disposition of original objects and items in a repository, as well as of the information objects relating to those originals [14]. Objects/items must contain the following three important features to be useful in a metadata-based system: what the object contains or is about (*content*), details of the features of the object (*context*), and formal set of associations within or among individual objects (*structure*) [14].

There exists several categorizations of metadata in literature based on the purpose/use. [15] considers three distinct categories. The first splits metadata into technical metadata (for system operations and maintenance) and business metadata (for valuation and interpretation). The second considers front-end metadata (for data storage and processing) and back-end metadata (for information delivery and use). A third category considers the functionality and shows the design and maintenance characteristics hidden by metadata.

Finally, in considering the use of metadata as a means of facilitating knowledge sharing in a web based environment, there is need of introducing technology components that make this possible. [16] brings out four key features to support this need. i) descriptive data structure standards for different kinds of community resource descriptions such as MARC, ii) markup languages and schemas for encoding metadata in machine-readable syntaxes, for example XML and RDF, iii) ontologies for semantic mediation between data standards such as CIDOC CRM, and iv) protocols for distributed search and metadata harvesting such as SOAP.

B. Metadata Search Process

Fundamental to the DDS is how the searching mechanism works, Figure 1 gives a general overview of the entire search process. The figure shows both the DDS layer and DHT layer.

The search process generally consists of three phases:

- *Initialization Phase*: The root node of the DDS is called by the requesting node.
- *Search Phase*: The broadcast message is sent to all the child nodes in the distributed structure.

- *Result Phase*: Each matching node sends a message to the requester.

Consider nodes A to E, which represent the logical nodes of the DDS, and the circle beneath represents the DHT-based P2P overlay. The dotted lines from the tree nodes intersecting the circle show how each logical node is mapped to the identifier space, thus having a responsible node on the DHT layer. It is possible to have one logical node mapping severally on the identifier space due to replication of data. The search process begins with the requestor node (R), a network participant starting a metadata search. R being a logical node maps to the identifier space, which enables the nodes that match the metadata keywords to reply back to the requester. Two nodes C and D that are marked black, represent nodes that match a search keyword with the dashed lines indicating a response message from these nodes to the requestor. More accurately the nodes that are responsible for the IDs of the matching logical nodes reply to the requestors ID. While the structure remains stable the responsible nodes may change due to churn. The solution builds in general on the idea of broadcast approaches known from metadata queries in P2P file sharing systems. Several solutions can be used to perform the metadata search. In this case, the solutions considered were LocalJoin, ParallelJoin, and BloomJoin, which are discussed shortly.

C. LocalJoin Approaches

LocalJoin approaches are so called because the join operation occurs at the local node. These are divided into *simple* and *parallel* LocalJoin.

1) Simple LocalJoin: Figure 2 shows the LocalJoin approach. Each figure is split into two parts by a horizontal line with the top section showing the sequential retrieval of the DDS and the bottom, the current state of a List-like data structure used throughout the whole algorithm. The algorithm proceeds as follows.

Step One: Take a case where we have two nodes in the DDS as in Figure 2a. The main node - marked with R is the root node that must be known in advance as an entry point for searching the DDS. To begin with, the root node copies all known ids of its children to a List-like data structure. To make the description simple, our example considers the root node having two children, therefore the list contains these two ids as entries. Having copied the unvisited ids to the list the root node starts with the first id from the list, $id1$, indicated by a black arrow pointing from R to $id1$, that is from parent node to child node. The child node being retrieved replies to the root and sends out a list of its unvisited child node ids back to the root. This reply is indicated by a dotted line pointing from the child node to the root node. The two boxes stand for the list of new unvisited child ids that are sent back to the root. At the end of the first step, the root node has visited the first id from the list and retrieved a list with new unvisited ids of its child.

Step Two: The search algorithm is as shown in Figure 2b. The process begins at R. Three nodes of the DDS are now visible. The first is the root node, R. The second node is the node that was contacted by the root in step one which is now colored black denoting that it has been visited. The connecting line between R and the id1 has no arrows as there is no communication between these nodes at the current step and it also symbolizes that this part of the tree is visited, hence already searched. The process of searching the third node $id2$ the approach is the same as in step one. R contacts $id2$ and obtain a list of new unvisited child node ids as a response. The arrowed line from R to $id2$ and the dotted line from $id2$ to R have the same meaning as in step one - query and response. At the List-like data structure, the first entry from step one is removed from the list. The pointer now moves to the next entry, that is, to $id2$, and the unvisited list of ids is appended at the end of the list.

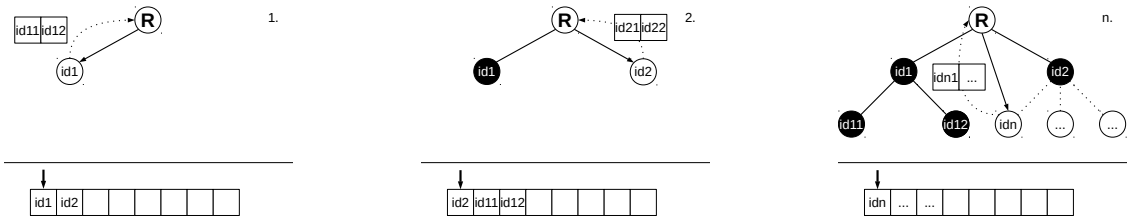
Step N***:*** Figure 2c portrays the nth step of the search process. There are now four black-colored nodes colored (visited and searched), and three white-colored nodes (not visited and searched). Dotted lines between node $id2$ and node idn and node $'...'$ show that the tree can have several more levels between these two nodes and in addition, it can have an arbitrary structure. The binary structure is chosen only to demonstrate the algorithmic steps. The same applies for the left subtree starting from the root node. Here also an arbitrary tree structure is more realistic than in the given example. Now supposing that part of the DDS is visited and searched (black nodes) and part of it is not (white nodes) as indicated in Figure 2c. The root node contacts the next id from the List-like data structure, for example, idn, which replies with a list of unvisited child node ids. If it has no children, the return value might be an empty list. The arrowed and dotted lines stand for query and reply as in steps one and two. From the List-like data structure, the first entry is removed and the pointer moves to the next id, idn in this case. Previously received ids are appended to the list as symbolized by the three dots. The algorithm terminates when no ids are left in the List-like data structure. Since the leaf nodes do not add new ids to the List-like data structure and the pointer moves forward in each step, clearly, then this algorithm terminates.

From a theoretical point of view, we know that the runtime of this algorithm follows a linear growth given the number of nodes of the DDS. Important to note is that different implementations of this data structure leads to a different behavior of the algorithm. For example, a stack-like implementation results into a Depth First Search algorithm.

D. Parallel LocalJoin

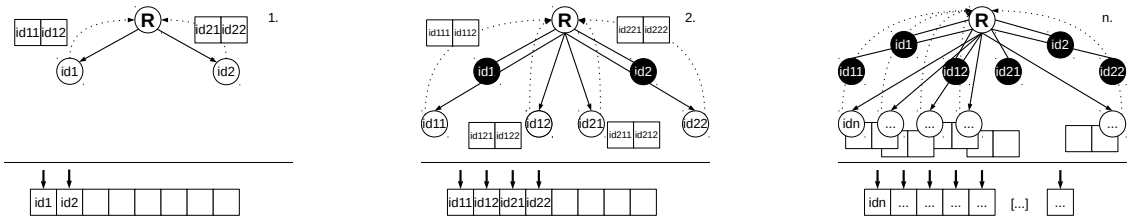
Parallel LocalJoin improves the idea of Simple LocalJoin by making use of the distributed nature of the network, that is, all unvisited ids are contacted in parallel in each step, returning a new, composite list of unvisited ids. Figure 3 illustrates the Parallel LocalJoin algorithm. The algorithm proceeds as follows:

Step One: Figure 3a shows this step. Similarly, as in the Simple LocalJoin, the root node, R, needs to be known in advance for the search process to begin. In comparison to



(a) Step One (b) Step Two (c) Step N

Fig. 2. Simple LocalJoin



(a) Step One (b) Step Two (c) Step N

Fig. 3. Parallel LocalJoin

Simple LocalJoin where nodes $id1$ and $id2$ were retrieved sequentially, in this case, the root node contacts both in parallel, shown by the two arrows pointing from root the node to child nodes. Both nodes then reply with their list of subsequent child nodes. Node $id1$ replies by sending a list of $id11$ and $id12$, while node $id2$ replies with its child nodes $id21$ and $id22$. As above, the binary structure is used as an example and the DDS in the application can be arbitrary. Also R does not get replies at the same time but rather waits as long as the longest reply for R to receive every list of unvisited ids from its children. The List-like data structure at the bottom of the figure shows the idea of parallelization. Two arrows point at the entries $id1$ and $id2$ meaning the algorithm processes both entries in this step. At the end of step, one R has a list of new unvisited ids from its children.

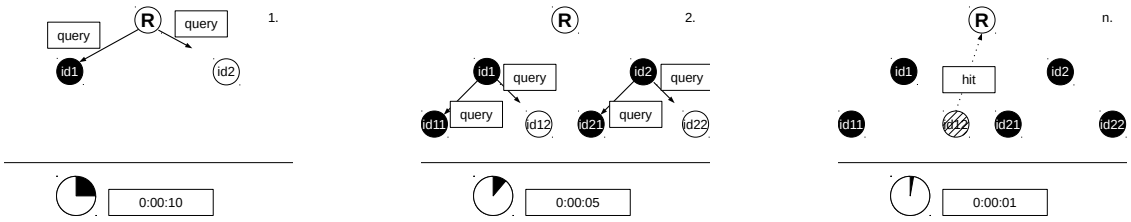
Step Two: As illustrated in Figure 3b, R first removes the visited ids from the List-like data structure and adds all new ids to it, in this case, $id11$ to $id22$. The connection lines and black-colored nodes $id1$ and $id2$ mean the algorithm processes this part of the DDS. R then, in parallel, contacts all ids from the List-like data structure, indicated by arrowed lines, and receives a list of child nodes (new unvisited ids) from each contacted node.

Step N***:*** Figure 3c considers the nth step. A few assumptions are made. Firstis that the search algorithm processed part of the DDS, indicated by black nodes and connections from them to the root. Also, the List-like data structure contains further entries, starting with idn and continuing arbitrarily, implying that R still needs to contact these ids to receive new ids of possibly unsearched nodes. The replies are indicated by dotted arrows from the white unvisited nodes to R. At the level of the data-structure now an arbitrary number of entries needs to be contacted in parallel. The runtime of the algorithm depends on the access time of the list. For simplification, we do not consider this access time when calculating the time it took to search DDS. Instead, we consider the duration of sending and receiving messages and the duration of retrieving DDS nodes. The algorithm stops when no new ids are left and the list is empty.

E. Network Join Approaches

1) Asynchronous NetworkJoin: The main idea of Asynchronous NetworkJoin is that the root node gives control to its child nodes to perform the search. They in return forward the search to their children in a flooding manner and so forth. Another important aspect of this approach is the timer that is discussed shortly. The Figure 4 shows the process undertaken in the asynchronous NetworkJoin. Similar to Simple LocalJoin the figures shown are divided into two regions by a horizontal line. While the top part shows involved DDS nodes and messages sent between them, the lower level shows the timer which is necessary for the this algorithm. It proceeds as follows.

Step One: Consider Figure 4a which shows this step. It begins with the root node, R, which must be known at the beginning of the algorithm. R sends messages to each of its children, indicated by arrow-headed lines from node R to nodes $id1$ and $id2$. The assumption is that R root has only two child nodes, but in reality, it can have more than two children. This binary structure of the tree is used for demonstration only. The right arrow is shown to stop half way between node R and node $id2$. This is used to show that the messages arrive asynchronously. This means that we cannot tell which of the child nodes of the root gets the message first. The message type that is sent from R to its children is a `query` message. Node $id1$ is marked black indicating it already received a `query` message and its payload does not



(a) Step One (b) Step Two (c) Step N

Fig. 4. Asynchronous NetworkJoin

contain meta-data that matches the `query`. In contrast node $id2$ is colored white which means it has not been visited and therefore there is no information whether it matches the search criteria. The bottom part shows a countdown timer initiated to a fixed time of 10 seconds in the example. The timer indicates that it is not possible for R to tell if the whole tree structure has been searched. Therefore it waits a maximum of 10 seconds after which the search terminates with the result that it may not have traversed the whole tree.

Step Two: Figure 4b gives a pictorial of how this step proceeds. R has no connection to nodes $id1$ and $id2$ because they are not communicating with R. As in Figure 4a nodes $id1$ and $id2$ send `query` messages to their respective child nodes. The half-way arrows also indicate the asynchronous arrival of the messages. Also, node $id2$ may start to query its children before node $id1$'. The different coloring has the same meaning as in Figure 4a - black denotes visited and not matching the `query` while white shows the node is not visited. At this moment the timer may be half way through and will continue to count down unless a match occurs.

Step N***:*** Figure 4c shows a possible outcome of ***Step Two***. The node $id12$ was searched and matches the `query`, indicated by the black stripes. It then sends a `Hit` message to the root node which is shown by the dotted line. When R receives such a `Hit` message it stops the timer and the algorithm changes its status from `in progress` to `finished successful`. A possible different outcome not illustrated in Figure 4c is the count down reaching zero without a match occurring. The status becomes `finished with the unknown result` as there may be nodes yet to be visited and searched. The algorithm may be improved as follows: when a `Hit` message reaches R, it resets the timer and waits for possible further `Hit` messages. The timer can be reset adaptively to the time it took for a first match to occur or can be fixed.

F. BloomJoin Approaches

1) BloomJoin: BloomJoin utilizes remote method invocation (RMI), that is, in each step more participants are involved in searching a rising amount of ids and the root node accesses the search results at each step. Figure 5 illustrates the steps undertaken in performing the BloomJoin algorithm which is explained as follows.

Step One: In Figure 5a, the root node, R, is the only participant involved in the search. R invokes itself to perform a search step on its own immediate child nodes. Therefore the R contacts its children and receives access to the participants. At the end of this step R has access to its child nodes, can remotely invoke them to search their children and can compare their payload to look for matching nodes regarding the search criteria.

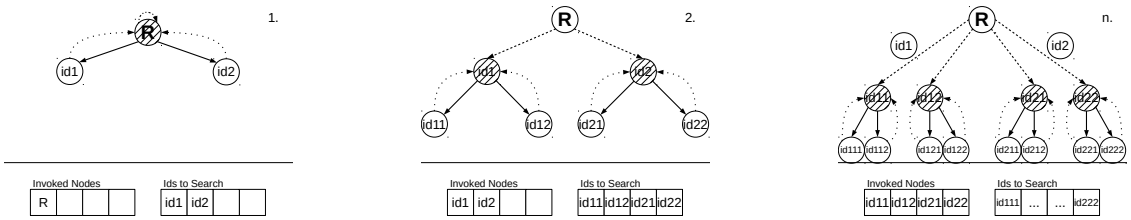
Step Two: This is shown in Figure 5b. R remotely invokes the child nodes, that is, nodes $id1$ and node $id2$ requiring them to perform searches with their respective children. Although R initiates this step, nodes $id1$ and $id2$ are those who carry the network load and calculation in this step. Like R, both $id1$ and $id2$ contact their respective children, forwarding access to them back to R. Finally, $id1$ and $id2$ will have received access to nodes $id1$, $id2$, $id11$, $id12$, $id21$ and $id22$, checked if any of these nodes matches search criteria and then passed access to them back to R.

Step N***:*** The last step in Figure 5c portrays the process to an infinite number of levels. Again R initiates the start of the search step remotely, while nodes $id11$ through to $id22$ search and gain access to further nodes. The dashed line represents a RMI while the arrowed and dotted lines represent the contact of the parent to the child node and the access response.

III. Evaluation Setup

In evaluating the quality and costs of the search mechanisms in DDS, six metrics are used.

- **Bandwidth (*Traffic*)**: Consideration is on the network load for each algorithm. The amount of traffic generated for each search process in addition to the traffic to keep alive the DHT overlay is therefore the focus. The measurement unit is either kilobyte or megabyte per second (Kbps/Mbps), depending on the type of algorithm type
- **Operation Duration (*Time*)**: How long it takes for each algorithm to search through the whole DDS or to find the first match for a given search query is the focus.
- **Hop Count (*#Hop*)**: This is the number of times a system message is forwarded during the search phase.
- **Reliability Ratio (*Ratio*)**: Measures the reliability of the search algorithms. It gives the ratio of the total number of successful operations during each search process against the total number of started operations. The aim is for a success ratio of 100% for all search operations.



(a) Step One (b) Step Two (c) Step N

Fig. 5. BloomJoin

- **Number of Messages (*#Msg*)**: Calculates the system load using the average number of messages sent in the network during the search phase.
- **Number of Contacted Nodes (*#Nodes*)**: Desired is the average number of nodes contacted for each of the search algorithms

The search algorithms are implemented and evaluated the search algorithms in the event-based simulator Peerfact-Sim.KOM [17], [18], [19] which provides for the analysis using the proposed metrics. The simulator provides various p2p layers, such as the application layer, service layer, data management layer, overlay layer, as well as several networking layers. The DHT simulated is based on Chord [20] along with the DDS and measurements of retrievability, retrieval delay and traffic overhead were also done. Additionally, the DDS List, Set and tree were implemented in the data management layer, which also runs an implementation of PAST [21] as replication service. The networking model used was the global network positioning (GNP) model [22]. The experimental scenarios also considered churn, based on measurements in KAD [23] for a more realistic simulation environment. In the setup, a network consisting of 1000 nodes is generated and simulated for 180 minutes. The size of each node is also specified as 4 kilobytes, for a total size of 4 megabytes for the DDS. The overlay joining process for all nodes occurs from minute 0 to 30. At the 50th minute mark, one node begins building a DDS tree structure and stores it in the DHT. Searching commences at the 120th minute and continues till the end of the simulation period for all search algorithms. Furthermore, we distinguish two outcomes of the algorithms. In order to gain a wider range of insights in the behavior of the search algorithms, three DDS trees were implemented and evaluated, the binary tree, deep tree and customized broad tree. On each of the three DDS types a group of 1000 participants execute one of the four existing search algorithms: i) LocalJoin (LJ), ii) Parallel LocalJoin (LJP), iii) NetworkJoin (NJ), and iv) BloomJoin (BJ).

A. Scenario 1: Binary Tree

The binary tree structure is configured to ensure that every node except the leaf node has two children nodes in the tree. The participants create a balanced binary tree with 1024 node, that is, 7 levels. Figure 6a shows an example of such a binary tree. The dotted lines indicate that the tree has more levels.

B. Scenario 2: Deep Tree

The deep tree structure is implemented with 1024 entries, in which every node except the leaf node has a single child node in the tree. Figure 6b is a deep tree. As in the binary tree, the dotted lines indicate more levels until the leaf node is reached. The number of levels for the deep tree equals the total number of nodes, and in this case, 1024. The deep tree DDS also represents a list that maintains an ordered set of entries. A typical use may be in articles with connected comments on the timeline of a user. Every new comment can be attached to the previous last comment creating an ordered list of comments under the article. It can ,therefore ,be regarded as a prototype for List structures.

C. Scenario 3: Customized Broad Tree

An example of this is given in Figure 6c. The experimental setup considered a two-level deep tree structure. The DDS consisted of a root data node with a total of 1011 nodes that represent a set-like data structure. The first level has 10 nodes, each having 100 child nodes. Hence $1+10+(10\times100) = 1011$ nodes. The continuous dots indicate more nodes on each level than shown in the figure.

D. Exhaustive vs First Match Search

Two kinds of search techniques were implemented, which preceeded the join algorithms that were extensively evaluated.

1) *Exhaustive Search*: The algorithm proceeds through the entire DDS ensuring that it exhaustively searches every node.
2) *First Match*: Once the algorithm identifies that a match has occured, the search process stops.

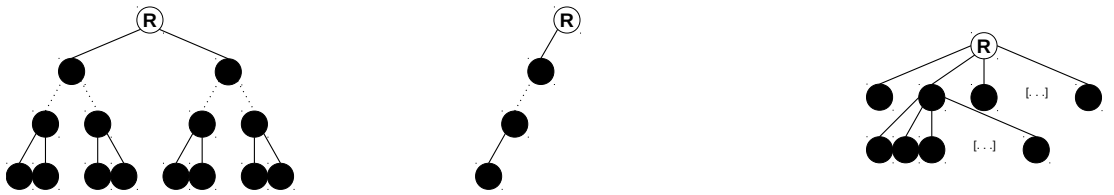
IV. Evaluation results

In this section, the systematic evaluation of the simulation results are presented, considering each of the two search mechanisms, exhaustive and first match searches.

A. Exhaustive Search

Table I are the metric results obtained for all the algorithms based on the exhaustive search run for the three DDS scenarios in consideration.

Bandwidth analysis results revealed that LocalJoin generated least traffic for binary tree and customized broad tree, while the NetworkJoin had least traffic for the deep tree



(a) Binary Tree (b) Deep Tree (c) Customized Broad Tree

Fig. 6. Tree Structure used in Experiments

	Binary Tree						Deep Tree						Customized Broad Tree					
Alg.	Traffic	Time	#Hops	Ratio	#Msg	#Nodes	Traffic	Time	#Hops	Ratio	#Msg	#Nodes	Traffic	Time	#Hop	Ratio	#Msg	#Nodes
LJ	2.3MBps	0.7s	5.84	0.9995	102K	14.07	3.9MBps	1.68s	5.76	1	140K	14.07	3.9MBps	2.1s	5.80	0.9979	235K	13.85
LJP	2.9MBps	6.18s	5.78	0.9989	143K	14.07	3.9MBps	1.66s	5.75	1	143K	14.07	7.1MBps	21s	3.15	0.8130	732K	14.01
NJ	4.1MBps	8.16s	2.74	0.9458	398K	1921	3.1MBps	0.75s	4.92	0.9990	163K	4242	7.6MBps	18s	2.24	0.8237	772K	912
BJ	2.5MBps	3.65s	5.47	0.9827	124K	14.07	3.4MBps	1.81s	5.72	1	127K	14.27	6.7MBps	19s	3.92	0.8270	699K	14.15

TABLE I
MEASUREMENTS FOR EXHAUSTIVE SEARCH

	Binary Tree						Deep Tree						Customized Broad Tree					
Alg.	Traffic	Time	#Hops	Ratio	#Msg	#Nodes	Traffic	Time	#Hops	Ratio	#Msg	#Nodes	Traffic	Time	#Hop	Ratio	#Msg	#Nodes
LJ	81KBps	0.17s	5.54	1	3K	14.27	440KBps	0.57s	5.80	1	13K	14.07	2.2MBps	0.66s	5.75	0.9995	98K	14.07
LJP	293KBps	0.55s	5.73	0.9988	14K	14.07	3.9MBps	1.65s	5.74	0.9999	44K	14.06	7.3MBps	21s	3.35	0.8129	753K	14.21
NJ	4.4MBps	5s	3.37	0.9735	392K	3206	521KBps	0.25s	4.90	1	25K	859	7.1MBps	17s	1.72	0.8302	722K	948
BJ	82KBps	0.12s	5.72	1	3K	14.07	440KBps	0.47s	5.79	1	13K	14.07	7.4MBps	20s	3.06	0.8147	752K	14.04

TABLE II
MEASUREMENTS FOR FIRST MATCH

structure. In terms of *operation duration*, LocalJoin utilized the least amount of time for binary tree and customized broad tree while the NetworkJoin algorithm used the least duration when the deep tree was used. The *number of hops* needed for the all the DDS structures was lowest when the NetworkJoin was used. Interestingly, the *reliability ratio* for all four join algorithms was above operations 0.9 for all the DDS structures. The LocalJoin had best values for reliability when the binary tree and customized broad tree were analyzed. For the deep tree, the ratios for all join algorithms were optimal except for the NetworkJoin. As for the *number of messages* the network generates, the LocalJoin algorithm again shows the lowest value in the case of all three DDSs. Finally, the *number of contacted nodes* is average at 14 nodes for all the algorithms except the NetworkJoin algorithm, where the count is in the hundreds.

Clearly, using the exhaustive search method along with the various join algorithms in comparison, the best algorithm, even with churn considered, is the LocalJoin algorithm. However, the NetworkJoin algorithm seems to show some promise when the DDS being considered is the deep tree, if the reliability ratio, number of messages generated and number of nodes contacted are not considered. But the fact that there is a high number of nodes contacted by the NetworkJoin algorithm may be an advantage if there is a need to perform regular updates in the system. However, the traffic generated is a cost that the design has to consider carefully.

B. First Match Search

Table II gives the summary for the first match search. As for the exhaustive search analysis, this analysis considers the optimal performing algorithm for each metric. Considering *bandwidth*, lowest traffic is generated by the LocalJoin algorithm for the binary tree and the customized broad tree. The LocalJoin and BloomJoin tie the lowest traffic when the deep tree is considered. The operation duration of the LocalJoin is lowest when the binary tree and customized broad tree are the DDSs if LocalJoin is used. For the deep tree, time is lowest for the NetworkJoin. *Number of hops* is lowest when NetworkJoin is used for all DDSs.

The *reliability ratio* is observed to be optimum in the binary tree when LocalJoin and BloomJoin algorithms are used. For the deep tree, all join algorithms except the Parallel LocalJoin algorithm give optimal ratios. The customized broad tree realizes best ratio value when the LocalJoin is used. The *number of messages* in the network is lowest for the binary tree when BloomJoin is used. The deep tree achieves the least number of messages for the LocalJoin and BloomJoin while the customized broad tree achieves minimal messages if LocalJoin is the join algorithm. Finally, for the *number of nodes contacted*, as for the exhaustive search, all algorithms have an average of 14 nodes that are contacted to complete search except the NetworkJoin algorithm.

Just like the exhaustive search, using the first match search with the various join algorithms revealed that the most consistent algorithm was the LocalJoin algorithm. The NetworkJoin algorithm is the poorest performer although when the deep tree is taken into consideration, there are certain performance parameters (specifically, operation duration, number of hops and reliability ratio) that are seen to be better than in the other join algorithms. The BloomJoin comes in a close second to

the LocalJoin when all parameters are under scrutiny for all trees.

C. Results Summary

In this section we presented an in-depth analysis of the four search implementations regarding traffic duration, hop count, correctness, number of contacted nodes and number of messages.

NetworkJoin reduces the average hop count to almost 0.5 during the search. The LocalJoin algorithm is the best algorithm with regards to all specified evaluation metrics. It is followed closely by the BloomJoin algorithmin terms of traffic, and also it has a better duration when considering the first match search experiments for the binary and deep tree. The worst performance is observed with the NetworkJoin algorithm when all metrics are evaluated together.

A limitation of the algorithms becomes apparent when handling broad tree structures. In previous experiments with a single level broad tree with 1000 child nodes, which models a set, three out of four algorithms failed to terminate, LocalJoin was the only exception. After customization of the broad tree to a two-level tree, maintaining the total DDT size at 1000 nodes (see section III) the algorithms performed better. Even with the correctness ratio at about 82% for LocalJoinParallel, NetworkJoin, and BloomJoin shows a weakness on a broad tree topology. In addition, NetworkJoin opens room for further improvement.

One major improvement is to limit the number of forwarded messages by checking the recipient of the query messages. Another suggested improvement is adjustment of the timer mechanism to make more specific statements about the termination result of the algorithm. Finally, another recommendation is to use a fitting algorithm in case of first match use cases as the results vary most and it is possible to find an optimal search algorithm for each tree topology depending on which metric needs to be optimized. For exhaustive search, the algorithms perform similarly so that there is no single favorite. Overall LocalJoin seems to respond best to both scenarios and the majority of DDT topologies.

V. Conclusions

Overlay and P2P networks such as Distributed Hash Tables are more and more important for modern applications, because of their various advantages. Distributed Hash Tables are commonly used in sophistical applications such as online social networks because it offers a self-organizing, scalable and robust solution to store and searches for the single data element. Searching metadata is an essential topic for modern applications, therefore, in this work, search mechanisms that support metadata-based searches in DHT overlays are introduced and evaluated.

References

[1] S. Buchegger and A. Datta, "A case for P2P infrastructure for social networks-opportunities & challenges," in *Proceedings of the Sixth International Conference on Wireless On-Demand Network Systems and Services, 2009. WONS 2009.* IEEE, 2009, pp. 161–168.

[2] S. Joseph and T. Hoshiai, "Decentralized meta-data strategies: Effective peer-to-peer search," *IEICE Transactions on Communications*, vol. 86, no. 6, pp. 1740–1753, 2003.

[3] S. Joseph, *P2P MetaData Search Layers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 101–112.

[4] J. Callan, "Distributed information retrieval," *Advances in information retrieval*, pp. 127–150, 2002.

[5] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Commun. ACM*, vol. 46, no. 2, pp. 43–48, feb 2003.

[6] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: Search methods," *Computer Networks*, vol. 50, no. 17, pp. 3485–3521, 2006.

[7] R. Akbarinia, E. Pacitti, and P. Valduriez, "Data currency in replicated DHTs," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 211–222.

[8] C.-F. M. Chan and S.-H. G. Chan, "Distributed hash tables: Design and applications," in *Handbook of Peer-to-Peer Networking*. Springer, 2010, pp. 257–280.

[9] S. Dahan, L. Philippe, and J.-M. Nicod, "The distributed spanning tree structure," *IEEE transactions on parallel and distributed systems*, vol. 20, no. 12, pp. 1738–1751, 2009.

[10] K. Bok, J. Kim, and J. Yoo, "Cooperative caching for efficient data search in mobile p2p networks," *Wireless Personal Communications*, vol. 97, no. 3, pp. 4087–4109, 2017.

[11] A. Koz and R. I. L. Lagendijk, "Distributed content based video identification in peer-to-peer networks: Requirements and solutions," *IEEE Transactions on Multimedia*, vol. 19, no. 3, pp. 475–491, 2017.

[12] R. Al-Aaridhi and K. Graffi, "Sets, lists and trees: Distributed data structures on distributed hash tables," in *IEEE IPCCC: Proceedings of the 35th IEEE International Performance Computing and Communications Conference IPCCC*, 2016.

[13] R. Al-Aaridhi, A. Yueksektepe, T. Amft, and K. Graffi, "Distributed Data Structures Improvement for Collective Retrieval Time," in *IEEE WPMC: Proceedings of the 19th International Symposium on Wireless Personal Multimedia Communications*, 2016.

[14] A. J. Gilliland, "Setting the stage," *Introduction to metadata*, vol. 3, 2008.

[15] G. Shankaranarayanan and A. Even, "The metadata enigma," *Communications of the ACM*, vol. 49, no. 2, pp. 88–94, 2006.

[16] T. Gill, "Metadata and the web," *Introduction to metadata*, vol. 3, pp. 20–38, 2008.

[17] M. Feldotto and K. Graffi, "Comparative Evaluation of Peer-to-Peer Systems Using PeerfactSim.KOM," in *IEEE HPCS'13: Proceedings of the International Conference on High Performance Computing and Simulation*, 2013.

[18] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator - Experiences and Lessons Learned," in *IEEE P2P '11: Proceedings of the International Conference on Peer-to-Peer Computing*, 2011.

[19] A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, and R. Steinmetz, "PeerfactSim.KOM - A Simulator for Large-Scale Peer-to-Peer Networks," TU Darmstadt, Tech. Rep. Tr-2006-06, 2006.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *SIGCOMM '01: Proc. of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001.

[21] P. Druschel and A. I. T. Rowstron, "PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility," in *IEEE HotOS '01: Proc. of the Workshop on Hot Topics in Operating Systems*, 2001.

[22] T. Ng, S. Eugene, and H. Zhang, "Global Network Positioning: A New Approach to Network Distance Prediction," *ACM SIGCOMM Computer Communications Review*, vol. 32, no. 1, 2002.

[23] M. Steiner, T. En-Najjary, and E. Biersack, "Long Term Study of Peer Behavior in the KAD DHT," *IEEE/ACM Transactions on Networking*, vol. 17, 2009.