Sets, Lists and Trees: Distributed Data Structures on Distributed Hash Tables

Raed Al-Aaridhi and Kalman Graffi
Department of Computer Science, University of Düsseldorf, Germany
Email: alaaridhi@hhu.de, graffi@hhu.de

***Abstract*—Distributed hash tables and unstructured peer-to-peer overlay networks allow to store, search and retrieve single data elements. While this is useful for simple applications such as file sharing, for future sophisticated applications more sophisticated distributed data structures should be supported. In order to build, for example, architectures for distributed online social networks or distributed computing graphs, more advances data structures such as sets, lists of sorted trees should be supported by the peer-to-peer storage overlay with convenient access to the entries. Taking, for example, the set of albums in a distributed online social network. This set consists of a set of albums consist of sets of images, which themselves have each a list of comments. Using distributed data structures for sets, lists or trees in a distributed hash table allow to easily build sophisticated applications. In this paper, we present and evaluate a concept for such distributed data structures in peer-to-peer networks. In the evaluation, we show that all elements of each distributed data structure is successfully stored and retrieved and that the approach comes with low overhead and delay.**

I. Introduction

Peer-to-peer (p2p) networks were in the focus of interest during the phase of popular file sharing networks. Now concepts of the technologies are shifting towards sophisticated use cases for example in the field of high performance computing to organize data structures within compute centers, in the field of multimedia streaming as well as distributed online social networks. In specific, since the Arab Spring, the NSA scandal and the need for censorship-free, reliable communication platforms, p2p-based social networks are heavily discussed in academia.

Literature up to now focused on harnessing information on friendship-relationships for optimized distributed data storage, such as in [1], Safebook [2], DiDuSoNet [3] or FroDo [4]. Relying on friends, however, comes with the risk of lost data due to churn, as shown by Mega et al. [5]. As an application-independent solution, in PeerSoN [6] and OverSoc [7] a separation of overlay and data plane is proposed to be able to optimize the routing and the data handling independently. We proposed in LibreSocial [8], [9], [10] to use distributed data structures which benefit a wide set of potential applications, such as the mentioned online social networks, distributed service-oriented p2p-market places or for meta-data structures in the field of compute centers.

During the investigation for future p2p-based, storage-depending applications, one main limitation of DHTs became apparent: the bottleneck of single-object oriented data storage. The mentioned use cases are characterized by potentially large data structures. For example, an album set of a user in a social network might contain 20 albums with 40 pictures each, in addition, each picture might have a long list of comments, in total summing up to 800-2000 individual data entries. If only one single node is responsible for 2000 data entries, such as it could be the case in a single-object-based distributed hash table (DHT), that single node might be overloaded and the replication scheme might take to long to replicate all 2000 data elements to other nodes.

As related work, Dabek et al. proposed in [11], [12] the Cooperative File System (CFS) as a single user file system based on a p2p approach. However, CSF does not support multiple users. Athicha et al. presented Ivy in [13] as a read/write P2P file system which uses modification-logs to keep track of the changes on the data objects. In order to retrieve the latest version of an object, a starting copy of the object and all further modification-logs must be retrieved. This approach, while supporting multi-user modifications of single data objects, does not comply to the social networking scenario, where mainly multiple users edit their own data objects. Saman et al. proposed PeerGraph [14] as a data structure to store the video files while enabling efficient streaming. In [15] an XML store system is presented which aims at the storage of large XML documents which are split among several peers in the p2p system. Both approaches require knowledge about the internals of the data to be stored, which also limits these approaches for the storage of these kinds of data. We aim with our approach at a general DDS storage system.

Our approach termed Distributed Data Structure (DDS), is a solution for the proposed use case.

A DDS consists of data items in a DHT which contain

- a mandatory object ID under which this item is stored.
- optional pointers to further data items in the DDS graph
- and an optional payload, such as an image or comment.

Thus DDSs are pointer graphs in the DHT, in which the linked data objects contain ID information, pointers and payload. The ID can be created deterministically, e.g. as a hash of a clear identifier which is known to the retrieving nodes and thus can act as "entry point" to the DDS, or it can be completely random and only used for DDS internal purposes. With this structure, we are able to create various data structures, such as (unordered) sets, (ordered) lists, (unordered or sorted) trees or even flexible distributed data graphs. One

This work was partially supported by the DFG Grant GR4498 1-1 - OverlayMeter

whole DDS is thereby distributed across the network and each individual data entry/item can be easily replicated or retrieved in parallel. In order to retrieve the whole DDS, the collective of data items, each corresponding data item has to be obtained from the network. In our example, 2000 data items have to be looked up, which takes time, and retrieved, which can be done in parallel, and is thus saving time in comparison to a single data source.

The contributions of this paper are threefold:

- We introduce an approach for distributed data structures.
- Through simulations, we show the performance characteristics of our approach within extreme cases such as a 100-entry broad tree and 100-entry deep tree.
- We evaluate the distributed data structure in a social networking scenario with an album set storage use case.

In Section II, we present our concept of DDS and highlight their strengths. In this paper, we point our the quality of these DDS and present therefore in Section III the simulation setup and in Section IV the evaluation results.

II. Distributed Data Structures

DHTs support the storage and lookup of single data objects based on their identifiers (IDs). This can be supported by replication schemes, such as PAST [16]. For applications that demand complex and potentially large data structures, this single object focused view is a severe bottleneck, as it becomes unpractical to store large data structures, such as a list of comments or images in a single data object. To pick up the example of the photo album with 100 images in a p2p-based social network, storing these images on one node is disadvantageous in terms of reliability and efficiency. Upon the request for this album, the single node transfers the 100 images to the requester without any means of parallelization, thus with a low efficiency. The same applies for the replication, for which the set of images also have to be transferred regularly to other nodes to provide guarantees for availability. Nodes with popular content, which are requested frequently might be overloaded by these transfers.

DDS allow distributing the load by distributing the storage. The implementation of DDSs applies a self-managing storage layer, termed DDS layer, located between the actual application logic of the p2p-application and the DHT storage layer with replication support. Each data item in the DDS can then be stored using the DHT and is replicated by further replication schemes as regular DHT item. The DDS layer provides to the application layer useful interfaces to create, modify and retrieve popular data structures such as lists, sets or trees. It allows operating both on whole DDS but also on individual entries in such a DDS. For example, a user might call the interface of the DDS layer to create a new set of entries with given values, or to add a further entry, such as a comment entry, to the end of a DDS list. This is convenient to use, as it applies how data structures are used in the application. Often in p2p-based social networks, a new comment is added to a list of comments or a new image is added to a set of images. Thinking of DDS trees, distributed file systems or structured storage spaces can be created easily.

On the other side of the layer, the DDS layer uses the interfaces provided by the DHT and a replication scheme to deploy the DDS in the DHT. Therefore it stores the single data items of the DDS, containing the ID information, pointers and payload, as a regular data object in the DHT. Here, we assume that every single data item, that is stored in the DHT is also replicated. Approaches, such as PAST [16] can be used to provide data availability even under churn. In PAST, a data object is not only stored on the node responsible for its object ID in the DHT, but also on n further nodes that would become responsible the initially responsible $n-1$ nodes would go offline. Through local self-monitoring, this ratio of n replicas is monitored and adaptively maintained. Using the put/get-functionality of a DHT allows to store and retrieve single data items in the DHT. A DDS supports following functions:

Store DDS: Once a user wants to create a completely new DDS, such as a photo album to be stored in the p2p network, it calls the DDS layer's function to create a set, for example, with the selection of items or files to add. In addition, it is free to choose an identifier for this DDS or to leave it to the DDS layer. The DDS layer then computes the corresponding data structure, a set in this example, and each individual data item in it. Each image is put in a single data item, which have individual object IDs for the DHT. The selection of object IDs referring to these image data items is then added to a new data item, which mainly contains only these IDs as pointers and the ID information of the new object. Now the index data item, as well as each data item containing an image, is stored in the DHT and replicated there. Finally, the identifier or the entry point for the DDS, i.e. the ID of the data item that contains the top level indexing information for this DDS is reported back to the p2p application layer, so that this ID can be used further on to retrieve the DDS. In a case of failures during the storage of the DDSs, the failure is reported at this point of the process, allowing the p2p application layer to react on it.

For the p2p application layer, the process is convenient, as it simply calls a function to create a set of given images or files. Also it is convenient that the DDS layer does not interfere with the specific implementation of the DHT or replication, it just assumes that the put and get functions provided by the DHT are reliable and data in the DHT is replicated.

Retrieve DDS: For the retrieval of a DDS, the retrieving p2p application layer needs the object ID of the entry data item for this DDS. For lists this is the first entry, for trees, it is the root entry and for sets a specific indexing data item, which points to all set entries. In order to retrieve the DDS, the p2p application layer calls the DDS layer requesting to retrieve the DDS with this object ID. The DDS layer now iteratively retrieves entry by entry. With each entry of this DDS retrieved, potentially new pointers are obtained listing object IDs which also belong to this DDS. For the set of these pointers, that are to be resolved and the corresponding data items have to be retrieved, the DDS layer handles the parallel lookup and retrieval. Here one of the strengths of the DDS concept

enfolds, by retrieving parts of the DDS in parallel and being thus faster in the retrieval compared to the retrieval from one single node. Once all pointers are resolved and all data items corresponding to the requested DDS are retrieved, the DDS is handed over to the p2p application layer. In case of failures, the DDS layer is able to notice which pointer cannot be resolved or which data item is lost in the DHT. Nevertheless, the remaining parts of the DDS can be retrieved and a partly complete DDS can be handed to the p2p application layer. The faulty part can be highlighting within this DDS.

For the p2p application layer, this function is convenient, as it solely operates on data structures and does not need to consider networking issues. This promises an eased creation of novel, more sophisticated p2p applications. Also, it is convenient that the DHT and replication schemes are fully utilized, thus no functionality is doubled at other layers.

Modify DDS - Modify Values / Add Entry: As the main structure of DDS supports items to have both data and pointers, a concept which is used for inner nodes as well as leafs. Thus, the DDS in use can be analyzed by the p2p application layer to gain further insights into the internal structure of the DDS. It might be relevant for an application to operate more frequently on parts of a whole DDS, such as a sub-tree or the end of a list. Any DDS operation can also be performed on parts of the DDS using the object ID of this specific part of the DDS.

Now if the p2p application layer decides to modify the DDS, either by adding a new entry, e.g. at the end of the list, in a set or as a leaf in the tree, it can compute the corresponding data item that needs to be changed. For this single data item, it asks the DDS layer to replace it with a new data item with the modifications applied. The DDS layer the performs a new storage operation with the new data entry for the corresponding object ID of the data entry.

In several cases, the p2p application might not know the internal structure of the DDS and might not know the correct data item to modify. Therefore, for popular DDS, such as lists, trees and sets also typical functions, such as "append to list", "add to set" or "add as leaf", are supported. Here the p2p application layer only has to know the object ID of the overall DDS and the DDS layer then performs the corresponding lookups and actions at the right place in the DDS.

Modify DDS - Delete Entry: The complete or partial deletion of a DDS is closely related to the modification of the corresponding part of the DDS. Analogously, a deletion is an overwrite of the previous content with specific information, that calls the storing node to delete this data item and release the storage allocation. Thus, in case a single data item is requested to be deleted, the DDS layer has to locate it in the DDS, and request it to be overwritten by a special data item marking the deletion of this item. If the whole DDS is observable, we handle the pointers to this data item as well, depending on the data structure we look at. In a set, the pointer to the deleted data item has to be removed from the indexing data item. In a tree, the pointer to this data item, typically from its parent node to this entry, has to be removed as well. In lists, the pointer from the previous element has to be modified to point to the next element. While this works fine, when we have a complete overview on the DDS, it is challenging to handle when we operate only on a part of a DDS. Then, we might not be aware of all pointers pointing to this element.

In the case when we are not aware of all data items pointing at the specific data item to be deleted, it is more advisable to replace the data item to be deleted with a "tombstone" marking its deletion but maintaining the pointers. In [17] we elaborate on this special case in lists and show that in such a case, tombstones can be moved step by step towards the end of the list, where they can be finally deleted. In the remaining of this paper, we do not consider this special case but focus on the typical case where a simple deletion is possible.

Discussion of the Strengths and Weaknesses: The main strength of DDSs is that we are able to persist data on a DHT, among multiple users, providing classic data structures like lists, sets, trees or single objects. The main idea behind the implementation is that a set, list or tree is stored in the DHT as data items containing ID information, pointers and payload. The DDS layer, thus provides convenient functions for the higher level p2p application layer, which can use this to build more sophisticated p2p-based online social networks, data spaces or interactive applications, without having to bother with networking issues. Having the data items stored in parallel allows retrieving the data items from different nodes in a shorter time than from a single node, as the upload bandwidth of several nodes can be combined.

On the downside, having the data items separated over various nodes in the network requires time to lookup the individual positions of the data and to iteratively retrieve the data. This might induce a high delay and many lookups in comparison to one single, often quick lookup in the case when the data structure would be stored on one single node.

In the evaluation of this DDS concept, we, therefore, show first, that all collectives of data items creating a list, set or tree are retrieved and thus the DDS is always retrieved successfully, even under churn. The retrievability of the DDS is the essential metric for the quality of our approach. In addition, we have a look at the costs induced by the pointer structure in terms of lookup delay for the whole DDS as well as traffic load induced on the nodes. In the following, we first present our evaluation setup to obtain this information in Section III and discuss then the results in Section IV.

III. Evaluation Setup

In the evaluation of the quality and costs of the DDS, we focused on three main metrics.

- Storage and retrieval success: We compare the number of storage/retrieval requests for DDSs with the number of successful storage and retrieval operations. In specific, we aim at a 100% success ratio for both operations, as they are essential for the use of DDS for any application.
- Retrieval delay: The storage and retrieval of lists, sets and trees are reflected at corresponding operations on a set of linked data items. We evaluate the time needed to deploy and retrieve DDSs in relation to their size.

- Traffic overhead / number of contacted nodes: As DDS are stored distributedly, we are interested in the overhead in terms of contacted nodes, induces lookups and traffic with respect to individual storage or retrieval calls.

In order obtain these metrics, we implemented and evaluated the DDS in the event-based simulator PeerfactSim.KOM [18], [19], [20]. PeerfactSim.KOM offers to simulate various p2p layers, such as a p2p application layer, service layer, data management layer, overlay layer, as well as several networking layers. We simulated the DDS on top of Chord [21] and measured the retrievability, the retrieval delay and traffic overhead. We implemented the DDS lists, set and tree in the data management layer, which also runs an implementation of PAST [16] as replication service. For the networking model we used the global network positioning (GNP) model as stated in [22]. In our scenarios, we modelled churn based on measurements in KAD [23] for a realistic environment.

We performed tests both with and without churn to measure the impact of churn. In both cases a maximum of 1000 nodes was active. In the begin of the simulations, from minute 0 to minute 40, a total of 1000 nodes joined the network. In simulations without churn, these nodes remained active until the end of the simulation at minute 240. In simulations with churn, first the nodes remained all in the network from minute 40 to minute 120. At minute 120 KAD-based churn started, resulting in a drop of nodes to around 650 at minute 240.

To gain a wide range of insights in the behaviour of DDSs, we conducted the investigation in four scenarios reflecting the three DDS categories lists, sets and trees. The tree is thereby a superset of a list and a set. A deep tree, which is restricted to one child entry per inner entry, is a list, as a clear order of the entries is maintained. A broad tree with one root data item and pointers to all leaf data items is thereby the prototype of a set. Thus, by analyzing various forms of trees, namely deep trees, broad trees and random trees, we get a complete view on the DDSs we consider in this paper. These various forms of trees have been evaluated in the following four scenarios.

A. Scenario 1: Album Example

In order to demonstrate the feasibility of the DDS, we present and discuss a small example. We consider the use case of the p2p-based online social networks, such as LibreSocial [24], [25], which uses profiles, albums and friends lists and therefore would benefit of DDS. In specific, we consider a set of album with one album, this album stores two images. The DDS structure is as follows, described with the tuple (Object ID, pointers, payload).

- Album-Set item: ID=hash(Album-Set), Pointers=(Album-1, Album-2), Payload=(AlbumSetCover.jpg)
- Album-1 item: ID=hash(Album-1), Pointers=(Image-1, Image-2), Payload=(Album-1-Cover.jpg)
- Image-1 item: ID=hash(Image-1), Pointers=(), Payload=(Image-1.jpg)
- Image-2 item: ID=hash(Image-2), Pointers=(), Payload=(Image-2.jpg)

This DDS is created and stored by one node in the network at the time t=50 minutes. The time t=55 minutes, another random node performs a DDS Retrieve operation for this DDS by initiating an Item Retrieve for its "entry" item, being the Album-Set Item identified by its ID "hash(Album-Set)". Once the item of the Album-Set is retrieved, the querying node retrieves the item related to the pointer in the Album-Set, namely the item stored under the ID "hash(Album-1)". As the item of Album-1 is retrieved, the pointers are further followed and the items with the IDs "hash(Image-1)" and "hash(Image-2)" are retrieved. When the complete DDS is obtained and all pointers are resolved, the result, i.e. the Album-Set, is delivered to the p2p application layer at the querying node. Besides the performance of a complete storage and retrieval of the DDS, we are further interested in the performance of a Modification Operation for a single entry. For this, Image 2 is modified and retrieved again. This operation is more efficient as only a single entry is retrieved, modified and stored instead of a whole selection of entries as shown in the Fig. 1a.

B. Scenario 2: Deep Tree

In this scenario, we present a deep tree structure with 100 entries, in which every node except the leaf node has one single child node in the tree. This deep tree DDS also represents a list, which maintains an ordered set of entries. Typical use cases for such DDS are for example 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.

In the simulations, at t=60 minutes a node creates and stores this DDS in the network. In the time from t=70 minutes to t=130 minutes this single DDS is retrieved by 250 nodes four times each, i.e. 1000 times, in the network. With this behaviour we want to mimic an interest group for that content. In this scenario, we are interested in the reliability of our approach in specific under churn. The chain of pointers must not break in order to obtain the complete DDS. Thus, we expect a complete retrievability shown by 100% of successful store operations, as well as DDS retrieve and item retrieve operations.

C. Scenario 3: Broad Tree

For this use case, a tree structure is built which is 3 levels deep. The DDS consists of a root data item which points at 50 further data items. One of these 50 data items at depth 2, has itself pointers to further 50 data items at depth 3. Thus in total, we have 1 data item at depth 1, 50 data items at depth 2 and 50 data items at depth 3. The example represents e.g. a photo album set of a user. The root data item points at 50 photo albums of a user, where the first photo album points at 50 images. Here as well, at t=60 minutes one node creates and stores this DDS in the network. In the time t=70 minutes to t=170 minutes the DDS is retrieved by 250 nodes four times each, i.e. 1000 times, in the network. Also here, we want to mimic an interest group for the content. In the setup of this scenario, we are interested in the benefits that we obtain through the parallelized processing of the retrieve calls for single data items in the DDS. We expect a DDS retrieval

delay close to the consecutive retrieval delay of 3 data items, while the load is distributed to 101 uploading nodes.

D. Scenario 4: Average Behavior

In this scenario, we investigate the impact of having many DDS in the network. In specific, we consider a network with 1000 users / nodes, with each user storing a DDS with 56 data items and further on retrieving DDS of the other users. The DDS consist of one single data item for the album set of the user. The album set points at 5 albums with each album having 10 photos. Thus a DDS tree of depth 3 is created with 56 data items. In the time interval t=40 minutes to t=100 minutes, 200 nodes create their album set DDS with 56 data items and store it in the network. Starting from t=110m to t=170 minutes, a set of interested nodes retrieve a complete photo album DDS of another node. In total 200 DDS store and 200 retrieve operations are conducted in the simulation.

In this setup, we are interested in the average performance of the DDS, in specific with regard to the performance and costs related to an average use case. In specific, we are interested in the average delay and the load induced on the nodes.

E. Scenario 5: Flexible DDS Sizes

The previous scenarios tested a fixed number of store and retrieve operations per node in order to fully evaluate the performance and costs with a clear expectation. In this scenario, the nodes act more naturally and do not stick to a fixed number of store/retrieve operation but rather initiate a random number of such events from a flexible range. Here, we consider a network with 1000 nodes, with each node storing a DDS with at least 10 and up to 100 data items and further on retrieving DDS of the other users. In this Scenario, at t=60 minutes 200 node become active and create and store DDS in the network. In the time t=140 minutes to t=240 minutes the DDS are retrieved 200 times. In the setup of this Scenario, the nodes do not retrieve their own DDS.

IV. Evaluation Results

In this section, we present the evaluation results of our simulations. A detailed evaluation of the individual operations at the example of Scenario 1 is presented in Table I, which shows the basic procedure, performance and costs of individual operations. Table II shows the success ratio of the four main operations, namely Store, DDS Retrieve, Item Retrieve and Lookup in the various scenarios. The table clarifies that all operations are performed correctly even in large numbers. Finally, in Table III and Figure 1, we present the detailed analysis of the performance and costs in all scenarios. In specific, in Table III we present the average values, standard deviation and the maximum of the Hop Count, Store Duration, DDS and Item Retrieve Duration, as well as costs in terms of Forwarded Messages, Served Messages and Contacted Nodes with regard to all 5 scenarios as well as the distribution of these metrics in Figure 1 with all its subfigures.

TABLE I
Detailed Evaluation Results of Scenario 1

Step	Items	Depth of DDS	Traffic [byte]	Operation delay [ms]	Nodes contacted
DDS Store	4	3	3580	3535 ms	4
DDS Retrieve	4	3	3580	3482 ms	4
Modify Item	1	0	764	1149 ms	1
DDS Store	11	5	11027	6319 ms	11
DDS Retrieve	11	5	11027	5756 ms	11

A. Detailed Analysis of Scenario 1

In Table I, we present the performance and cost of all operations performed in the small Scenario 1. In the first and second step, the DDS Store and DDS Retrieve operation are performed, contacting both times 4 nodes and transferring the DDS with 3580 bytes. For the operations, in both cases around 3.5 seconds have been used, which result from the lookup of the 4 nodes as well as the data transfer to/from them. The third step, the modification of one item, requires the lookup of the corresponding node and the upload of a changed version of the item. This requires traffic of 764 bytes and a total time of 1149ms. This small example shows already the benefit of DDS, as individual modifications do only require the change of a small part of the DDS, which is traffic and time saving. Finally, in the fourth and fifth steps, a larger DDS is retrieved. This larger DDS is presented in Figure 1(a) in detail. Here 11 nodes are contacted and it takes approx. 6 seconds to lookup all nodes and store/retrieve the DDS. Here again, the initial lookup operations to find all 11 nodes as well as the subsequent transfer of the data sum up. The evaluation shows that all operations are performed as assumed and that we have again in terms of delay and traffic when only individual items in a DDS are modified.

B. Test of Functional Requirements: Success Ratio

In this part of the evaluation, we focus on the success ratio of all operations in larger setups. Table II shows that all operations were conducted nearly to 100%. For those Store, DDS Retrieve, Item Retrieve and Lookup operations with nearly 100% success ratio, it means that only seldom failing peers or timeouts lead to single operation misses, resulting in a success ratio of 0.99*. There are two cases of lower success ratios, in Scenario 1 with and without churn, where only 96% of the lookup and even 94% of the store operations succeeded. However, this means just 1 out of 26 missed lookups which lead to 1 out of 16 unfinished store operation. In summary, the success ratio of the DDS operations is overall near 100% with

TABLE II
Operation Success Ratio in the DDS

Sc.	Stores		DDS Retrieves		Item Retrieves		Lookups	
	Init.	Ratio	Init.	Ratio	Init.	Ratio	Init.	Ratio
1n	16	0.938	5	1.00	26	0.962	26	0.962
1c	16	0.938	5	1.00	26	0.962	26	0.962
2n	510	1.000	1000	1.00	102000	0.998	102000	1.000
2c	510	1.000	1000	1.00	101025	0.999	101943	1.000
3n	510	1.000	1000	1.00	102000	0.998	102000	1.000
3c	510	1.000	1000	1.00	102000	0.999	102620	1.000
4n	12200	0.993	199	1.00	12139	0.998	12142	0.999
4c	12200	0.994	200	0.99	12135	0.999	12216	1.000
5n	12200	0.999	199	1.00	12139	0.999	12161	1.000
5c	12200	0.999	199	1.00	12139	0.999	12156	1.000

TABLE III
PERFORMANCE AND COSTS OF THE DDS SCHEME

Sc.	Hop Count			Store Duration			DDS Retrieve Duration			Item Retrieve Dur.			Forwarded Messages			Served Messages			Contacted Nodes		
	Avg.	σ	Max.	Avg.	σ	Max.	Avg.	σ	Max.	Avg.	σ	Max.	Avg.	σ	Max.	Avg.	σ	Max.	Avg.	σ	Max.
1n	5.92	1.17	6.75	0.64	0.51	1.00	3.13	0.18	3.25	0.64	0.16	0.75	2.64	1.90	7	2.78	2.11	5	20.00	21.21	35
1c	5.67	0.82	6.25	0.71	0.40	1.00	18.25	21.57	33.50	3.52	3.91	6.29	2.30	1.80	8	2.78	2.11	5	20.00	21.21	35
2n	5.91	0.06	6.07	0.46	0.00	0.46	130.38	12.78	215.50	0.83	0.12	1.61	561.96	436.84	2533	355.08	207.84	1750	408.04	6.50	510
2c	5.68	0.09	5.90	0.46	0.00	0.46	312.95	115.14	783.50	2.62	1.12	7.21	604.04	499.69	3418	272.95	236.86	1635	403.98	17.89	510
3n	5.86	0.06	6.03	4.28	0.00	4.28	19.20	1.28	28.75	6.31	0.39	7.42	561.98	444.89	2657	351.39	209.84	1743	408.02	6.51	510
3c	5.67	0.09	5.97	4.28	0.00	4.28	103.69	53.37	278.25	8.21	2.26	17.25	605.90	530.59	3001	288.72	242.83	1743	407.86	6.53	510
4n	5.84	0.20	6.33	0.73	0.18	1.61	7.60	4.43	47.00	1.21	0.24	2.56	59.04	47.94	289	13.23	12.65	85	121.63	4.32	122
4c	5.68	0.22	6.46	0.72	0.17	1.66	74.76	73.66	364.00	4.21	4.72	32.49	66.66	58.15	316	14.98	14.89	74	121.53	4.44	122
5n	5.75	0.21	6.30	0.75	0.22	1.77	10.68	11.05	126.00	1.37	0.36	3.41	72.53	57.85	397	16.29	16.02	122	121.61	4.33	122
5c	5.75	0.21	6.33	0.75	0.22	1.77	10.52	10.56	125.00	1.37	0.36	2.95	72.45	57.76	393	16.28	16.05	125	121.61	4.33	122

a few single failing operations that might happen. In cases where the numbers of store operation are big, e.g. Scenario 4, the success ratio is more than 99% for all operations.

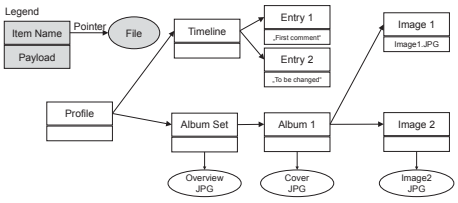
C. Detailed Analysis of Performance and Costs in Scenarios

In Figure 1 and Table III, we present a detailed analysis of the performance (Hop Count, Store Duration, DDS and Item Retrieve Duration) and costs (Forwarded Messages, Served Messages and Contacted Nodes) with regard to all 5 scenarios and all operations. The figures from Fig. 1(b) to Fig. 1(o) represent the distribution of the results over the peers for the 7 metrics and for all 5 scenarios. The setups without churn display the fundamental behaviour for DDS in P2P networks, while the setups with churn, in which a simple replication protocol is used to keep data available, gives us insights on the behaviour of DDS under strong network dynamics. Related to that, in Table III we summarized the average, standard deviation and maximum values of these metrics.

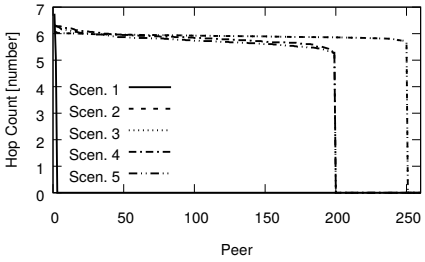
The Hop Count is for all scenarios between 5 and 6 as it shows in Table and Figure 1(b) to Figure 1(c). A Hop Count of around 6 is predictable in an overlay with 1000 nodes and is considered normal. The Store Duration is around 0.45-0.75 seconds for most cases, except for Scenario 3. The overall average storage time of around 0.5 seconds is defined through the hop count of 5-6 in combination with an average one hop delay of up to 100ms in the network model. Only for Scenario 3, we observe a special behaviour. Here the Store Duration reaches 4.28 seconds in average. The single broad DDS involves for the storage of 100 DDS items the creation and storage of this 100 items in parallel and this lead to getting a big bandwidth. The time consumption of this process is depicted in this increased Store Duration. Also in Figure 1(d) and Figure 1(g), we see that the Store Duration distribution in Scenarios 4 and 5 with many store operations is well balanced.

Next, we have a look at the DDS and the Item Retrieve Times, as they are tightly interconnected. As a DDS consists of several items, its overall retrieve performance highly depends on the Item Retrieve Time. In these Retrieve Times, the values differ most. In general the DDS and Item Retrieve Times are larger in cases of Churn. This is due to failed nodes that are still first contacted in the retrieval process. After a timeout, the next replication peer is tested, which results overall in a longer retrieve process in case of churn. In Scenario 2, the DDS Retrieve Time is largest, as we have have as DDS a Deep Tree Structure with its deep reference chain. For each retrieved DDS Item a further DDS lookup is initiated looking for the Item on the next level. The resulting average duration times of DDS Retrieves visualize the effect of deep DDS structures and the concept of recursive retrieval. The retrieval operation requires multiple lookups and round trip times adding together one after another. In Scenario 3 on the other hand, the broad DDS allows to parallelize Item Retrieve operations and thus lead to an improvement of the retrieval times. In the Figures 1(e) and 1(h), we present the distribution of DDS retrieval times, while in Figures 1(f) and 1(i), we present the Item Retrieval times. We observe, that the Item Retrieval times are very balanced over all scenarios, with the exception of Scenario 4 with churn. In Scenario 4, DDS of the size of 56 items is created. This is reflected in the figures when the retrieve duration increases in characteristics steps. For each step, another DDS was retrieved causing a bandwidth bottleneck at the retrieving node. The stepwise increase of item retrieval time is also observable in Figure 1(i) for Scenario 3 and 4 without churn, both scenarios reflect the retrieval of many items in parallel. Nevertheless, in nearly 100% of the cases the DDS have been retrieved successfully.

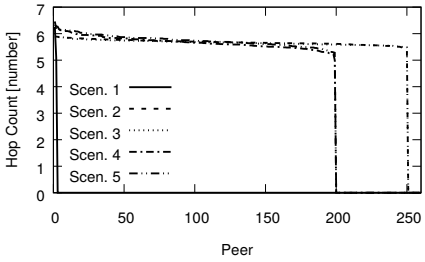
We measure the costs of the operation in terms of forwarded and served messages as well as the number of contacted nodes. The corresponding values are also presented in Table III and Figure 1(j) to Figure 1(o). The traffic for Scenario 1 is very low due to the small size of the DDS. Scenario 2 and 3 impose the deep and broad examples of a DDS and involve around 400 nodes to be contacted in order to store the DDS with 100 entries in a fully distributed fashion. Around 600 message are to be forwarded per node. This results for 1000 DDS retrieve operations, resulting in overall 102,000 Item Retrieve operations (see Table II). As each item retrieval requires a lookup of around 6, we have 612,000 messages to be forwarded by 1000 nodes, so 600 messages per node in average. The performance shifts between Scenario 2 and 3 in the case of churn. The deep DDS (Scenario 2) takes longer to be retrieved than the broad DDS (Scenario 3). With its deep reference chain, for each retrieved DDS Item, a further DHT lookup is initiated. In the case of churn, this lookup might fail and a timeout occurs. Through the aggregation of these timeouts and the recursive, non-parallel lookup behaviour, the performance of DDS is decreased in Scenario 2. While we also rather suggest harnessing the parallelization options in the DDS, i.e. through a broad DDS, also deep DDS are suitable



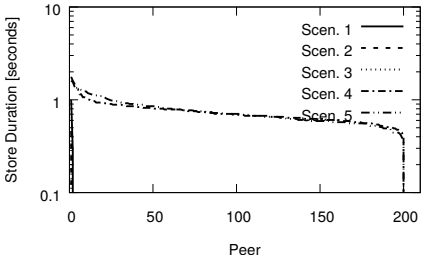
(a) Data Structure in Scenario 1



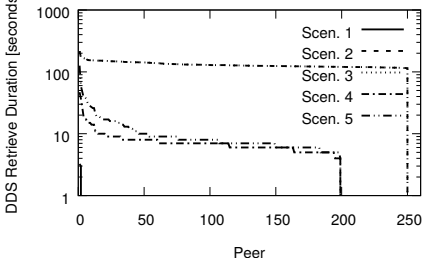
(b) No Churn: Hop Count



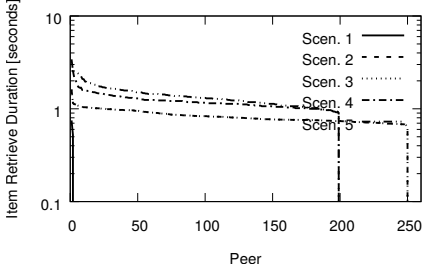
(c) Churn: Hop Count



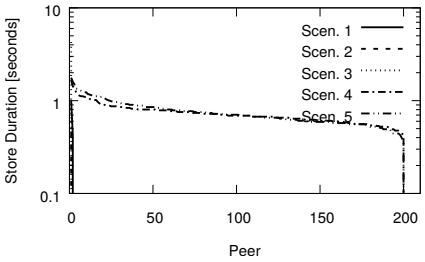
(d) No Churn: Store Duration



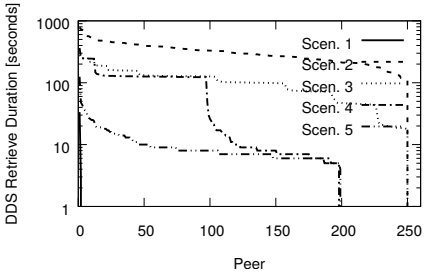
(e) No Churn: DDS Retrieve Duration



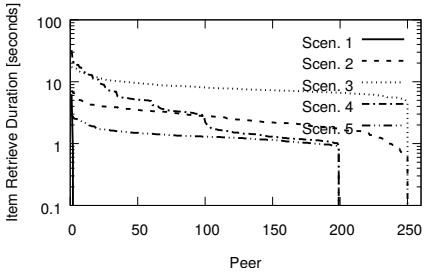
(f) No Churn: Single Data Item Retrieve Duration



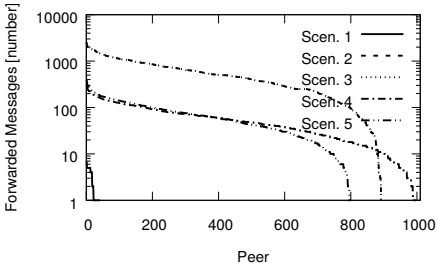
(g) Churn: Store Duration



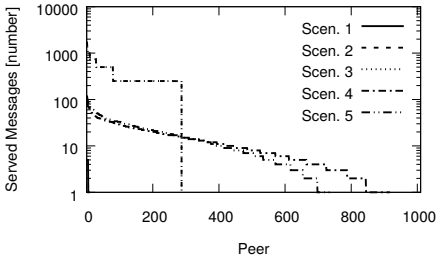
(h) Churn: DDS Retrieve Duration



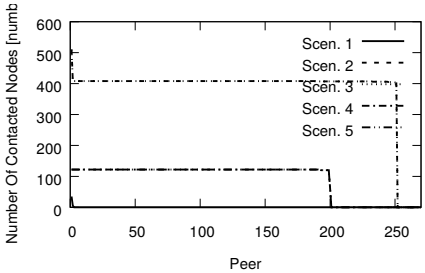
(i) Churn: Single Data Item Retrieve Duration



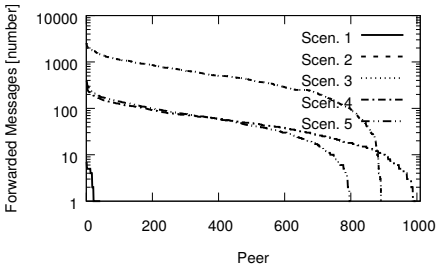
(j) No Churn: Forwarded Messages



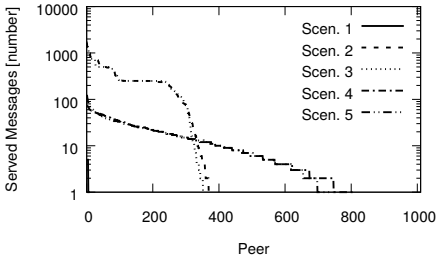
(k) No Churn: Served Messages



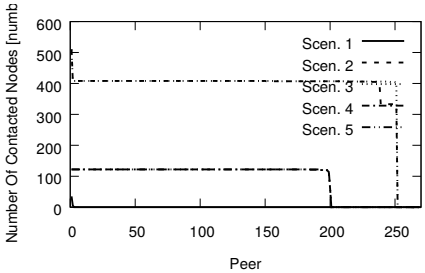
(l) No Churn: Number of Contacted Nodes



(m) Churn: Forwarded Messages



(n) Churn: Served Messages



(o) Churn: Number of Contacted Nodes

Fig. 1. Measurement Results with regard to the five Scenarios

if the items are very large and when the interaction takes place with individual items rather than the whole DDS. In Scenario 2 and 3 only 280 - 350 nodes are served, i.e. receive messages from store or item retrieve operations, in contrast to 700-830 nodes for Scenario 4 and 5. This is due to the single large DDS in Scenario 2 and 3. The large DDS and the low number of active nodes in Scenario 2 and 3, result in a higher number of Contacted Nodes per operation per node (Fig. 1(l) and 1(o)) and more Served Messages (Fig. 1(k) and 1(n)). Each operation in Scenario 2 and 3 involves due to the large DDS more lookups and thus more Forwarded Messages (Fig. 1(j) and 1(m)) than for Scenario 4 and 5. Due to the random behaviour in Scenario 4 and 5, the workload is more even, which is shown in the even distribution of the Forwarded and Served Messages. For all scenarios, the costs are in the setups with and without churn very similar.

V. Conclusions

P2P networks currently offer only storage/retrieve operations only based on single data items, although advanced applications, such as online social networks, would benefit from more advanced distributed data structures such as lists, sets or trees. In this paper, we propose a distributed storage scheme for Distributed Data Structures (DDS) which support these advanced data structures and DDS operations as well as individual operations on individual items in the DDS, such as selective modification. Our approach builds on a distributed pointer-structure which is stored in the DHT.

For the evaluation, we investigated the performance and costs in small and large scale scenarios, with and without churn, as well as on deep and broad distributed data structures in a network with 1000 nodes. In Table II, we have shown that all DDS operations are functional and resolved to nearly 100%. Small aberrations from this rules are caused by single defective lookups in the network. With this reliability, we can build various DDS types with their corresponding strengths and weaknesses. In Table I, we analyzed the performance of individual DDS operations and have shown in Figure 1 and Table III a detailed analysis of the performance and costs of DDS in 5 scenarios with and without churn.

While any kind of DDS benefits from the parallelization and the possible single item operations (such as for modification), nevertheless, DDS which are broad harness the strengths of a distributed data scheme better. Broad DDS are retrieved quicker, while deep DDS lack from an increased waiting time in unreliable p2p networks. We believe that our DDS approach will benefit the creation of further p2p-based applications.

References

[1] G. Mega, A. Montresor, and G. P. Picco, "Efficient Dissemination in Decentralized Social Networks," in *Proc. of IEEE Int. Conf. on Peer-to-Peer Computing (P2P)*, 2011.

[2] L. A. Cutillo, R. Molva, and M. Önen, "Safebook: A Distributed Privacy Preserving Online Social Network," in *Proc. of Int. Symp. on World of Wireless, Mobile and Multimedia Networks (WOWMOM)*, 2011.

[3] B. Guidi, T. Amft, A. De Salve, K. Graffi, and L. Ricci, "DiDuSoNet: A P2P Architecture for Distributed Dunbar-based Social Networks," *Peer-to-Peer Networking and Applications*, pp. 1–18, 2015.

[4] T. Amft, B. Guidi, K. Graffi, and L. Ricci, "FRoDO: Friendly Routing over Dunbar-based Overlays," in *IEEE LCN '15: Proc. of the IEEE Int. Conf. on Local Computing Networks*, 2015, pp. 1–9.

[5] G. Mega, A. Montresor, and G. P. Picco, "On Churn and Communication Delays in Social Overlays," in *Proc. of IEEE Int. Conf. on Peer-to-Peer Computing (P2P)*, 2012.

[6] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "PeerSoN: P2P Social Networking: Early Experiences and Insights," in *Proc. of ACM EuroSys Workshop on Social Network Systems (SNS)*, 2009.

[7] D. I. Wolinsky, P. S. Juste, P. O. Boykin, and R. J. O. Figueiredo, "OverSoc: Social Profile Based Overlays," in *Proc. of IEEE Int. Workshops on Enabling Techn.: Infrastr. for Collab. Enterprises (WETICE)*, 2010.

[8] K. Graffi, S. Podrajanski, P. Mukherjee, A. Kovacevic, and R. Steinmetz, "A Distributed Platform for Multimedia Communities," in *Proc. of IEEE Int. Symposium on Multimedia (ISM)*, 2008.

[9] K. Graffi, C. Gross, P. Mukherjee, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: A P2P-Based Platform for Secure Online Social Networks," in *Proc. of IEEE Int. Conf. on Peer-to-Peer Computing (P2P)*, 2010.

[10] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks," in *Proc. of IEEE Int. Consumer Communications and Networking Conf. (CCNC)*, 2011.

[11] F. Dabek, "A Cooperative File System," Master's thesis, Massachusetts Institute of Technology,Cambridge, Massachusetts, United States, 2001.

[12] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *ACM SIGOPS Operating Systems Review*, 2001.

[13] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: A Read/Write Peer-to-Peer File System," in *ACM SIGOPS Operating Systems Review*, 2001, pp. 31–44.

[14] A. S. Tosun and T. Korkmaz, "PeerGraph: A Distributed Data Structure for Peer-to-Peer Streaming," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, 2008.

[15] T. Thorn, A. Baumann, M. Fennestad, and P. Sestoft, "A Peer-to-Peer, Value-oriented XML Store," IT University of Copenhagen, IT University of Copenhagen, Denmark, Tech. Rep., 2003.

[16] 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.

[17] J. Janiuk, A. Mäcker, and K. Graffi, "Secure Distributed Data Structures for Peer-to-Peer-based Social Networks," in *IEEE CTS '14: Proceedings of the IEEE International Conference on Collaboration Technologies and Systems*, 2014, pp. 1–10.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[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.

[24] K. Graffi, C. Groß, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks," in *IEEE CCNC '11: Proceedings of the IEEE Consumer Communications and Networking Conference*, 2011.

[25] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz, "Practical Security in P2P-based Social Networks," in *IEEE LCN '09: Proceedings of the Annual Conference on Local Computer Networks*, 2009.