Adding Capacity-Aware Storage Indirection to Homogeneous Distributed Hash Tables

Philip Wette
Computer Networks Group
University of Paderborn, Germany
philip.wette@mail.upb.de

Kalman Graffi
Technology of Social Networks Group
University of Düsseldorf, Germany
graffi@cs.uni-duesseldorf.de

***Abstract*—Distributed hash tables are very versatile to use, as distributed storage is a desirable feature for various applications. Typical structured overlays like Chord, Pastry or Kademlia consider only homogeneous nodes with equal capacities, which does not resemble reality. In a practical use case, nodes might get overloaded by storing popular data. In this paper, we present a general approach to enable capacity awareness and load-balancing capability of homogeneous structured overlays. We introduce a hierarchical second structured overlay aside, which allows efficient capacity-based access on the nodes in the system as hosting mirrors. Simulation results show that the structured overlay is able to store various contents, such as of a social network, with only a negligible number of overloaded peers. Content, even if very popular, is hosted by easily findable capable peers. Thus, long-existing and well-evaluated overlays like Chord or Pastry can be used to create attractive DHT-based applications.**

I. Introduction

Peer-to-peer (p2p) overlays have been intensively researched in the last decade, mostly in the context of file sharing. With the rise of online social networks, decentralized alternatives are also being reviewed. Online social networks are the most popular web applications nowadays. Peer-to-peer based online social networks (P2P OSN) promise to connect the users, while preserving their privacy. P2P OSNs can be implemented through social topological overlays like Safebook [1], or by using proven peer-to-peer structured overlays and mechanisms, like PeerSon [2] or LifeSocial [3], [4]. In the latter, typical structured overlays like Chord, Pastry or Kademlia are used to host a social network. Prominent structured p2p overlays implementing a distributed hash table (DHT) consider each peer to have the same role and duty: i.e., being homogeneous nodes with equal capacities. Additionally, it is assumed that the objects stored in the DHT are uniformly distributed and thus no node is overloaded. As objects are neither equally requested nor of equal file size, each object produces different loads in terms of storage and bandwidth consumption. We face following two issues:

- Varying object *load*: object popularity is Zipf-distributed in the World Wide Web [5] and in social networks [6].
- Varying node capacities: node heterogeneity is strictly diverse in the OECD countries according to [7].

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901)

The critical capacity available at a peer p used to perform data provision is the *maximum upload data rate*, denoted as $v_{max}(p)$, which is the bottleneck rate at which data can be sent into the network. This rate can be determined by using various techniques, such as the one shown in [8]. We say a peer p is *overloaded* if $v_{max}(p) < \sum_{f \in F(p)} l_f$, where $F(p)$ are all files hosted at p and l_f is the average load file f produces.

We consider use cases in which large, popular files generate load going beyond the capacities of the majority of nodes in the network. While the separation of data indexing and storage might take the load of the indexing peer, the question occurs of how to pick the storage peers. Naturally, with a homogeneous approach and a random storage node selection, several nodes will be overloaded by the duty of hosting popular items, such as the photo album of a popular person in a P2P OSN. Additionally, P2P OSNs also introduce the requirement of data updates, which is typically not needed in file sharing applications. The data update requires that the location of each copy of the data items in the network is known. Thus, random caching becomes unusable. Also, virtual peers are not usable in this case, as shown in [9].

We address the question of capacity-aware storage node selection in DHTs: This paper presents an approach to easily make homogeneous DHTs capacity aware. The approach itself is non-invasive and can be deployed incrementally: i.e., peers which want to participate can be active, but not all nodes have to be. We present the application of our approach on Chord [10] as a show case, turning it into Adaptive Chord. As a result, we show by simulation that the ratio of overloaded peers and the average data retrieval delay is drastically lowered.

Next, in Section II, we discuss related work to approach the issue of overloaded peers in homogeneous p2p overlays. Subsequently, Section III presents our approach for a general extension of homogeneous DHTs at the show case of Chord. We present the concept of a parallel capacity overlay and reason the use of capacity-based mirroring. A thorough evaluation of our approach is given in Section IV, where we show that the modified Chord version is robust under Churn and lowers the ratio of overloaded nodes drastically in a social networking simulation scenario. Section V concludes that the general approach presented in this paper is useful for building quality-aware novel p2p applications.

II. Related Work

A DHT provides key-value storage and a lookup function. The stored value can be either a data object, in which case it is called *direct storage*, or a pointer to a data object, which is called *indirect storage*. In the case of direct storage, several index shifting approaches have been proposed to balance the load over all peers. To this end, various interpretations of a "load-balanced network" can be found in the literature. Several more theoretical papers address load balancing in terms of assigning each peer an equal *number* of files, a file count proportional to the peers' capacity or proportional to the peers' consumption. However, in practice, with highly heterogeneous file sizes and popularities, the load experienced at a peer is *not* linked to the number of files it offers. In contrast to the existing work in this field, this paper aims at assigning files to peers such that no file generates more load than the assigned peer is able to offer.

Existing load-balancing techniques for structured overlay networks can be grouped into three main areas: usage of *virtual peers*, *repositioning of peers* and the *repositioning of files* in the identifier space. An overview of these approaches is given in [11]. When using the technique of virtual peers, as in [12], [13], each physical peer creates a number of virtual peers, each with a unique identifier. Through its virtual peers, a physical peer is located at different positions in the identifier space at the same time. The more resources a peer offers, the more virtual peers it creates. Thus, the expected number of files is proportinal to each peer's resources. In [12] and [14] each peer creates a number of virtual peers proportional to their available capacities. If the load a virtual peer has to carry increases beyond tolerance, the virtual peer can be moved from one physical peer to another. However, it remains unclear how to find the capable nodes that are able to host the virtual peer. The approach of altering the position of peers in the identifier space is applied in [15], [16].When ever the load of a peer gets too high, this peer shrinks its responsibility range in the DHT by changing its identifier. This approach has been adapted to Chord and Koorde in [17]. In [18] and in PAST [19] (file diversion) a file has k potential positions in the identifier space where only one of those is actually used, depending on the load the file generates and the resources present at the corresponding peers. PAST also uses replica diversion to find free capacites near a file. Beehive [20] adapts the prefix area in which a file is replicated, thus it adapts the load on the peers while allowing for mutable objects. In presented approaches share that popular files are repositioned in the network unaware of the capacities of the new storage peer. Thus, there is no clear strategy to find capable nodes. However, there exist capacity-aware solutions. Unfortunately they come with strong requirements on the overlay topology, like in [21] for hierarchical overlays or [22] for Cycloids. These approaches are therefore limited in their applicability; general concepts, like those presented in this paper, are favourable.

Obviously, it is advisable to separate data storage and indexing, to have more freedom in the data placement and to be able to use flat topologies. In this case, the storage node of a data item is reported to the indexing peer. The index peer is found via key-based routing (KBR [23]) interface of the DHT. The selection of the storage node can be either passive, meaning without a dedicated storage node selection strategy, or active. Passive storage node selection can be implemented as a cache. Unmanaged caches, meaning ones which are created during routing in the network like in [24], are not usable for our purpose, as data item updates or deletion requires the modification of the data at each storage node. Thus, the overview of the storage nodes needs to be maintained actively. The indexing node might act as a cache of the storage nodes. This cache can contain the list of nodes which downloaded the data item previously and might act as storage nodes in the future. While this approach is appealing, it does not guarantee that the files are hosted by capable peers.

We present an approach that addresses the real load induced by file objects in the DHT. Our approach shifts load from overloaded nodes to powerful nodes which we explicitly search and pick based on their capacities. Thus, the number of overloaded peers is reduced and the capacity of the peers is harnessed.

III. An Approach for Capacity Awareness and Load Balancing - Adaptive Chord

To reduce the number of overloaded peers in the DHT to a minimum, this section presents our approach of capacity-aware storage node selection in DHTs for indirect storage. The approach aims at general applicability in various DHTs and requires only small modifications at the application level: i.e., it does not require the DHTs themselves to be modified. Thus, a DHT can be incrementally extended, node by node, with our approach. Although the approach is general for DHTs, we present it exemplary at the show case of Chord. We name this capacity-aware Chord version *Adaptive Chord*. Our approach consists of two main concepts: *object mirrors* and a *capacity overlay*.

We introduce a new role for peers that we call *supplier* and apply a separation of the data indexing and storage. A peer p supplies a file f if p provides a copy of f for download. We call such a copy of a file a *mirror*. This is very distinct from a peer q being responsible for f due to $f.id \in [q.predecessor.id, q.id)$.

Figure 1 shows how a file is retrieved by using suppliers. We assume the identifier $f.id$ of the file f is known to the requester. First, the peer p that is responsible for f is looked up in the DHT by using the DHT's lookup function. Then, if p does *not* supply f by itself, p redirects the query to the supplier of f which finally transmits f's data to the requester. Note that only the node responsible for the data object (p) is aware of the mirror supplier. If p does not participate in the capacity-aware protocol enhancement and supplies f by itself, f's data is transmitted directly from p to the requester.

The concept of object suppliers allows us to decouple the responsibility of a data object from the object's load. But the main question remains: *Where to place the mirrors in the network?* Ideally, the mirror suppliers should be picked based

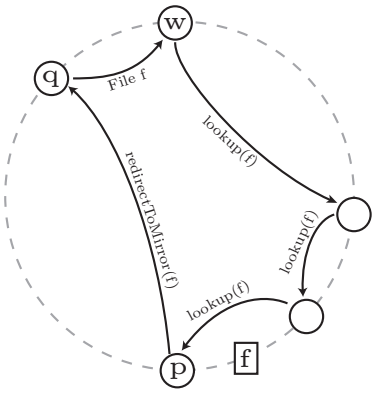


Fig. 1. Locating Data Objects in Supplier-enhanced DHT

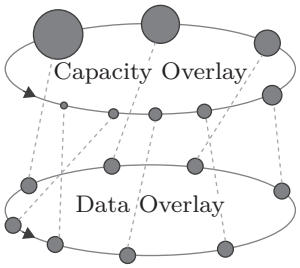


Fig. 2. The Two Layers of Adaptive Chord

on their capacities and the capacity requirements of the data objects. To solve the issue of supplier selection, we introduce a new, additional overlay that can be used to address peers by their free capacity, called the *capacity overlay*. This new capacity overlay as well as its interaction with the data overlay is presented next. To make the presentation easier we refer to the concrete DHT Chord and its capacity-aware variant, Adaptive Chord.

A. Overlay Architecture

The Adaptive Chord overlay consists of two parallel, independent sub-overlays, which are illustrated in Figure 2. The capacity of each peer is indicated by size: the larger the circle, the more capacity is present at the peer. At the bottom you see the *data overlay*, which is a Chord network that is used for a) defining which peer is responsible for what file, b) locating the peer that is responsible for a particular file, and c) repairing the structure of the network in the presence of churn. In this network, peers are arranged in ascending order by their unique identifier, which is chosen as a hash value of their corresponding IP address and is therefore pseudo random. At the top you see the *capacity overlay*. The capacity overlay is also a Chord network but with the special property that peers are *not* arranged based on their identifier but on the amount of their free capacities. For a peer, participating in the Adaptive Chord network means to participate both in the data overlay *and* in the capacity overlay. Peers that voluntarily participate in the capacity overlay share their free capacities and are available for being picked as suppliers.

1) Data Overlay: For the incremental deployability of the approach we consider that the nodes in data overlay, such as Chord, are unmodified. For nodes that wish to participate in Adaptive Chord, which is capacity aware and overload avoiding, the nodes in the data overlay are additionally equipped with the following functions: a) Each peer collects information about each file it is responsible for, such that the mean amount of traffic each file generates can be calculated; b) Each peer can host a number of mirrors for files it is not responsible for (in sense of the DHT). Whenever a peer receives a query for a file it has a mirror for, it answers the query by transmitting the file to the requester; c) Whenever a peer p triggers another peer q to host a mirror for file f, p transfers the file to q. From now on, q hosts a mirror for f; d) If a peer p receives a query for a file f it is responsible for and there exists a peer q that hosts a mirror for f, p redirects the query to q.

2) Capacity Overlay: The capacity overlay is a second DHT with a one-dimensional identifier space, in our show case a Chord network; it is used to efficiently locate peers based on their free capacities. To do so, we define identifiers such that a peer p that has more free resources available than a peer q will have a higher identifier than q. If identifier selection is based on this strategy, the Chord protocol will automatically sort the peers in ascending order by its free resources.

In the capacity overlay, identifier selection is done by Equation 1:

$$ID = \min\left\{2^m - 1, (2^m - 1) \cdot \frac{v}{M}\right\} \tag{1}$$

where m is the length (in bits) of the identifiers, v the amount of free resources the peer offers, and M a globally used huge constant, chosen such that it is unlikely that $\frac{v}{M} > 1$.

As Equation 1 maps an amount of free resources to an identifier, we can easily find a peer with at least v free resources by calculating the ID of a peer with exactly v free resources and then lookup the peer in the capacity overlay that is responsible for that ID, which is the peer with the next higher identifier and thus has the next higher free resource count.

In Adaptive Chord we are using the resource *free available upload data rate* (in Kb/sec), which is the rate at which a peer is able to send additional data into the network. For the design of the capacity overlay, we assume the *last mile model* [25] for determining the available data rate between two nodes in the network. This is a common hypothesis when looking at p2p networks formed by home computers accessing the Internet through DSL or ISDN lines. Throughout this paper v_{max} will describe the maximum rate at which a peer is able to send data into the network. We additionally assume that very weak peers do not participate in the capacity overlay. They would introduce maintenance overhead, but their contribution in hosting mirrors would be useless.

B. Protocols in the Data Overlay

Load in a peer-to-peer network strongly depends on file popularity which is changing over time. Thus, it might happen that at some point in time a peer p, responsible for a file f, is no longer able to process all requests for f without being overloaded. In this case, a node not participating in the capacity overlay will be overloaded, while peers participating

```
1: load ← n.getMeanUploadSpeed()
2: remain ← v_max − load
3: if remain < κ then
4:     if n.MirrorCount > 0 then
5:         n.removeMirrorWithHighestLoadAndInfromOrigin()
6:     else
7:         File f ← n.getFileWithHighestLoad()
8:         Peer q ← n.findPeerWithFreeLoad(f.getLoad())
9:         if q ≠ NULL then
10:            q.addMirror(f)
11:            n.fileOutsourcedTo(f, q)
12:        end if
13:    end if
14: else
15:    threshold ← (remain − κ) · (1 − γ)
16:    if ∃ File m ∈ outsourcedFiles : m.load < threshold then
17:        Peer q ← m.getMirrorHoster()
18:        q.removeMirror(m)
19:        n.gotFileBack(m)
20:    end if
21: end if
```

Fig. 3. Algorithm n.LoadBalancing()

additionally in the capacity overlay are able to find a supplier for files they cannot serve sufficiently. Having found a fitting supplier using the capacity overlay, the corresponding file is mirrored to the new supplier and subsequent queries for the file are forwarded to the mirror. In the following, we present the protocols that have to be applied by the nodes in the data overlay when participating in the capacity overlay.

1) File Redirection: The nodes in the DHT must support redirection. Thus, the peer responsible for a file knows the supplier of that file and redirects the query to a corresponding supplier in the case that the file has been mirrored previously. The supplier will then transfer the files contents to the requester.

2) File Access Statistics: To be able to find a peer suitable to supply a certain file, the peers need to know the traffic load caused by the individual files. To obtain this information, each peer holds a list of timestamps for the last r requests for each file. From this list, together with the file size, the average traffic caused by the file can be determined. Since each request for a file f is routed from the responsible peer p to the supplying peer q, both p and q can calculate the traffic caused by f.

3) Load Balancing: The core of Adaptive Chord is its load-balancing, overload-avoiding algorithm which is used to create and tear down mirrors depending on the actual load situation at a peer. The algorithm can be seen in Figure 3 and consists of three operations: a) local mirror removal, b) mirror creation and c) remote mirror removal.

Local Mirror Removal. Peer p is considered as overloaded if it has less than κ resources available. Choosing κ as $\min\left(70Kb/sec, \frac{1}{5} \cdot\ v_{max}(p)\right)$, for instance, would ensure that a) files are at least transferred at 70 Kb/sec, and b) the participants (which we assume to be home computers) still have some upload capacities for other applications. If peer p is overloaded and supplies one or more mirrors, p removes the mirror for file f that is generating the most load. This is done by contacting the peer responsible for f and requesting it to supply the file by its own again. At this point, p no longer serves requests for f and thus load on p has been lowered.

Mirror Creation. If peer p is overloaded and does not supply any files it is not responsible for, p can create mirrors on other peers. To do so, it selects the file f that generates the most load l. Then, a peer q is looked up in the capacity overlay that has at least l free resources. This is done by calculating an ID i, corresponding to l with Equation 1. q can now be found by looking up the peer responsible for i in the capacity overlay. In a last step, f is sent to q which now supplies f in the network.

Remote Mirror Removal. To lower the overall mirror count in the network, mirrors for a file are removed if the peer responsible for that file is able to provide the required resources on its own again. To this end, each peer p checks for each mirrored file f, if p can provide the file on its own, without being overloaded again. If such a file exists, p removes the mirror for f and answers queries on f directly by transmitting the file.

C. Protocols for the Capacity Overlay

In our approach we propose to use a second p2p overlay in parallel to the data storage overlay. Routing in this second overlay allows to find peers with given capacities to use them as suppliers. In order to implement such an overlay, the following protocols must be applied.

1) Join: Identifiers in the capacity overlay are computed from the free resources of the peers. To circumvent equal identifiers in the capacity overlay, we partition the identifier space of the capacity overlay in equally sized partitions, each representing one unit of resources. Each of these partitions consists of $\frac{2^m-1}{M}$ identifiers. Peer identifiers in the capacity overlay are computed by first calculating the corresponding partition and then selecting a random position within this partition. If this position is already in use the peer selects the next higher free position. The join algorithm for the capacity overlay can be seen in Figure 4.

```
1: n.id ← (2^m − 1) · v/M + rand(0, (2^m−1)/M)
2: while p.FindSuccessor(n.id -1).id = n.id do
3:     n.id ← n.id + 1;
4: end while
5: successor ← p.FindSuccessor(n.id)
6: predecessor ← NULL
```

Fig. 4. Algorithm n.Join(p) to Join the Capacity Overlay on Peer p.

2) Update Position: Since the node identifiers in the capacity overlay represent the *free* resources of each peer, identifiers in the capacity overlay are dynamic. Whenever the free resources of a peer change, it has to update its identifier and thus update its successor, predecessor and finger entries in the routing table. This process can be seen as a) leaving the network, b) calculating a new identifier, and c) joining the network again. This process leads to a high fluctuation that can be seen as synthetic churn occurring in addition to the natural churn due to hosts going online and offline. As churn has a negative effect on the performance of the network, we want to keep the churn as low as possible. To lower the synthetic churn, peers only update their position in the capacity overlay if their free resources have changed by more than ϵ. We decided to set $\epsilon = 0.1 \cdot v_{max}(p)$ to prevent slightly changing request patterns to cause synthetic churn.

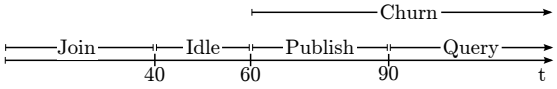


Fig. 5. Schedule of the Simulation Actions

D. Discussions

In the presented approach, there are some variants and design decisions to discuss. Firstly, how to handle peers with low maximum upload data rate. For a file f that is rarely requested, the average traffic l_f is low. Consequently this file could be located at a peer with a low maximum available upload data rate without overloading this peer. This of course leads to low data rates when accessing this file. To prevent peers with low bandwidth from supplying files in the network we use the parameter κ such that all peers p with $v_{max}(p) < \kappa$ are categorized as being overloaded. As a consequence these peers aim to not provide any files for download at all.

The more free resources a peer is capable of, the faster files can be distributed by this peer. To decrease the average download time it makes sense to mirror files on peers that have a high free resource count. However, to be more churn resilient, peers hosting a large amount of mirrors have to be avoided. To distribute the load on a larger amount of peers with more capacities, we introduce a threshold ω which marks the minimum amount of free resources a new supplier has to have. ω is variable and calculated by each peer individually. Each time a peer successfully creates a mirror, ω is raised by 20%. If mirror creating failed, ω is lowered by 20%.

The setting of the parameters, especially κ, γ, ω and ϵ, depends on the scenario and the capabilities of the peers. Here, rather than having constants, relative values and ratios corresponding to average and minimum capacities in the network seem desirable. For that, monitoring approaches like PushSum [26] or SkyEye [27] can be used to obtain a global view on the capacities of the nodes in the network and to adapt the parameters in relation to them.

IV. Evaluation

The evaluation was conducted with simulations to allow a timely and detailed investigation of the effects of a second capacity overlay. For our simulations we used the discrete and event-based p2p simulator *PeerfactSim.KOM* [28], [29], [30]. We implemented Adaptive Chord and conducted simulations in a network with 1000 nodes both with and without churn over 24 simulated hours.

Peers were simulated to be located all over the world with Internet connection characteristics being distributed like determined by the OECD Bandwidth Report from 2007 [31]. The delay between the peers was determined by GNP [32]. Chord was implemented according to [33]. The file sizes of the shared files in our simulation are equally distributed from 0 to 10 Megabyte and the popularity patterns of the files follow a Zipf distribution with $p(x) \sim (1/Rank(x))^{0.6}$. These assumptions are characteristic for various use cases, like WWW documents or personal information in social networks.

Name	Inter query time	Pop. shift	Churn
busy_unstable_nochurn	10 min.	10 min.	no
idle_unstable_nochurn	30 min.	10 min.	no
busy_stable_nochurn	10 min.	60 min.	no
idle_stable_nochurn	30 min.	60 min.	no
busy_unstable_churn	10 min.	10 min.	yes
idle_unstable_churn	30 min.	10 min.	yes
busy_stable_churn	10 min.	60 min.	yes
idle_stable_churn	30 min.	60 min.	yes

TABLE I
Scenario Setups

The schedule of each scenario is shown in Figure 5. Peers join the network within the first 40 minutes. We assume that all nodes participate in Adaptive Chord i.e., in the data and capacity overlay. For the following 20 minutes the peers are idling to ensure the stabilization process of the network has converged. In the next 30 minutes 10,000 files are published and afterwards queried until the 24 hour period ends. After the first hour, churn is enabled to simulate the fluctuation that is present in large distributed p2p networks. Files are transferred using TCP/IP.

A. Scenarios

To study the behaviour of Adaptive Chord we simulated different environmental scenarios. The scenarios are differing in the following parameters:

- **Inter query time:** Average time between two queries originating from the same peer.
- **Inter popularity shift time:** Time between recalculation of the popularity of a file.
- **Churn:** Process of joining and leaving the p2p network.

To study the behavior of Adaptive Chord, we choose the inter-query time to be either 10 minutes, corresponding to a high load scenario, or 30 minutes, corresponding to a low load scenario. As over the lifetime of a file its popularity typically changes, we decided for an inter-popularity shift time of 10 and 60 minutes, respectively. Churn follows the Weibull distribution as learned by real-world observations [34]. The number of nodes in the network under churn is depicted in Figure 6(a). In the scenario with no churn, 1000 nodes are constantly online. A listing of all considered scenarios can be seen in Table I.

B. Parametrization

In our simulations we compare Adaptive Chord with a plain Chord implementation. For Chord, all stabilization methods are executed every 30 seconds by each peer having 3 backup entries for their successor.

For Adaptive Chord's capacity overlay the successor entry is checked each 10 seconds and one finger entry is updated every 20 seconds. Due to the high churn in the capacity overlay, each peer holds a list of 10 backup entries for its successor. The load-balancing algorithm was executed every 4 minutes by each peer. The data overlay is parametrized like the plain Chord network.

The minimum capacity for not being overloaded, κ, was set to $min(0.2 \cdot \ v_{max}, 70Kb/sec)$. The ratio of capacity that should remain free after canceling external mirrors, γ, was

Szenario	# O	v [Kb/s]	t [s]	σ t [s]	S [%]
busy_unstable_nochurn					
Chord	158.6	118.6	806.7	1471.9	71.8
Adaptive-Chord	18.2	275.4	85.9	154.0	97.1
idle_unstable_nochurn					
Chord	76.4	126.5	619.9	1152.8	87.7
Adaptive-Chord	5.3	243.8	72.8	108.1	98.8
busy_stable_nochurn					
Chord	160.4	116.2	810.2	1467	72.2
Adaptive-Chord	18.9	258.2	87.5	156.3	97.0
idle_stable_nochurn					
Chord	75.1	133.2	611.8	1143.4	87.8
Adaptive-Chord	5.3	229.9	73.9	110.1	98.8
busy_unstable_churn					
Chord	29.4	249.2	514.4	958.4	38.1
Adaptive-Chord	21.8	379.8	186.1	493.5	43.6
idle_unstable_churn					
Chord	12.7	207.8	370.3	577.9	43.2
Adaptive-Chord	11.0	333.7	159.1	340.1	46.8
busy_stable_churn					
Chord	30.3	231.7	511.6	966.6	39.4
Adaptive-Chord	24.5	352.8	185.3	480.3	44.4
idle_stable_churn					
Chord	11.8	216.5	321.8	509.7	43.2
Adaptive-Chord	12.9	323.7	194.5	423.4	45.6

TABLE II
SIMULATION RESULTS: #*O*: AVERAGE NUMBER OF OVERLOADED PEERS. *v*: AVERAGE DOWNLOAD SPEED. *t*: AVERAGE DOWNLOAD TIME. σ *t*: STANDARD DEVIATION OF DOWNLOAD TIME. *S* SUCCESSFUL DOWNLOAD RATIO.

chosen to be 0.2. ϵ was set to $0.1 \cdot v_{max}(p)$. The minimum capacity for a supplier, ω, was initially set to 150 Kb/sec for all peers, however, it varies per peer over time.

C. Metrics

Since in a p2p network, performance is best measured from the user's perspective, the metrics used in our simulation are: the average time to finish a download and the data rate at which files are transferred. In addition we keep track of the number of overloaded peers, where a peer is considered to be overloaded if its free upload capacities are less than κ. To model the impatience of the users, a file transfer is canceled if its average data rate drops below 1 Kb/sec. We call such a download *unsuccessful*.

D. Results

Table II summarizes the results of our experiments. For selected discussions of individual simulation runs we refer to the results from the scenarios with busy queries (every 10 minute per peer) and unstable popularity, which changes every 10 minutes. Our observations are summarized next.

1) Number of Overloaded Peers: The main goal of our approach presented with Adaptive Chord was to significantly reduce the number of overloaded peers and thus to increase the download speed. In Adaptive Chord the number of overloaded peers is in any case smaller than in Chord with a maximum of 24.5 in Adaptive Chord and 160.4 in plain Chord, (#*O* in Table II). In Adaptive Chord, 24.5 peers are overloaded at maximum, often much smaller. Churn increases the number of overloaded peers, as less suitable suppliers are available. The table shows that a frequent (busy) inter-query time induces more load and thus also more overloaded peers.

Figure 6(b) plots the number of overloaded peers over time in a network without churn. After about 10 hours simulated time Adaptive Chord denotes only a negligible number of overloaded peers, where in Chord there are constantly more than 16% of the peers overloaded. Figure 6(c) illustrates the number of overloaded nodes in a network with churn. Since the load in the network is relatively low, the number of overloaded peers in both Adaptive Chord and Chord are low.

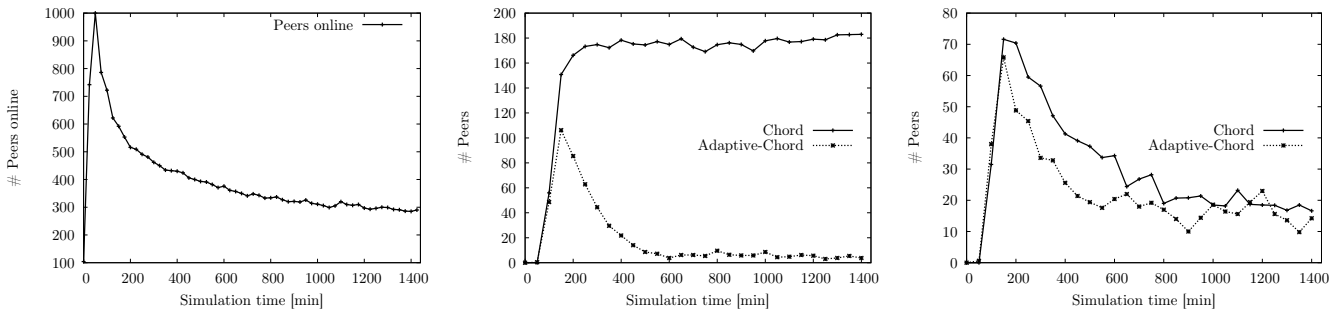
2) Download Speed: The metric v in Table II shows that the download speed in Adaptive Chord is in any case faster that in Chord. A performance gain of up to 232% is reached (in the scenario busy_unstable_nochurn). In plain Chord, the average download time is raised by the overloaded peers. Figure 7(b) depicts the corresponding download speed of Chord and Adaptive Chord in this scenario.

The number of online nodes over time in scenarios with churn is illustrated in Figure 6(a). By enabling churn, the performance of Adaptive Chord drops significantly, as we can see in Figure 8(b). This is mainly due to the fact that whenever a peer goes offline, all mirrors on this peer leave the network, too. The affected files have to be mirrored again on other peers. But as the process of mirroring a file also consumes capacities, the performance of the network drops when many mirrors are created. Nevertheless, using capacity-aware file placement as in Adaptive Chord improves the average download speed.

3) Download Time: The download time is relevant for users and depicted as metric t in Table II. Here, Adaptive Chord dominates the results by a factor. In scenarios without churn, the download time in Chord is a magnitude larger than in Adaptive Chord. This is due to the randomized supplier assignment in Chord.

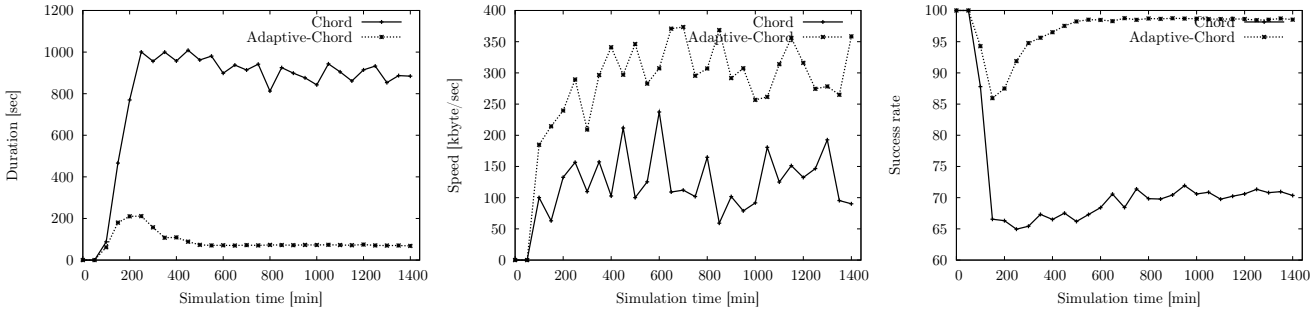
Figure 7(a) shows the average duration of a download in scenario busy_unstable_nochurn. It can be seen that in this scenario with high load and unstable popularity patterns Adaptive Chord is able to adaptively place mirrors on peers that are able to provide the required resources to serve file transfers at a good quality. In Adaptive Chord, download durations are up to nine times lower than in the plain Chord protocol. Figure 8(a) shows the average duration of a download for the scenario busy_unstable_churn. It can be seen that Adaptive Chord serves files about two times faster than Chord.

4) Success Ratio of Downloads: One critical metric for the rating of the p2p overlay is the success ratio of the downloads, which is depicted in the metric S in Table II. The table shows that in the case without churn, Adaptive Chord is able to serve more than 97% of all lookups and subsequent downloads. In Chord, however, the success ratio of the downloads is even in the case without churn only between 71% and 87%. Figure 7(c) visualizes the success ratio of downloads over time in busy and unstable scenario without churn. As Chord is only at 70% success ratio, Adaptive Chord serves downloads at over a 98% success ratio at the end of our experiment. It can be seen that the success ratio of downloads in scenarios with churn is lower, for that see Figure 8(c). This is due to the fact that the Chord implementations do not consider file replications, which thus leads to unavailable files and unsuccessful downloads



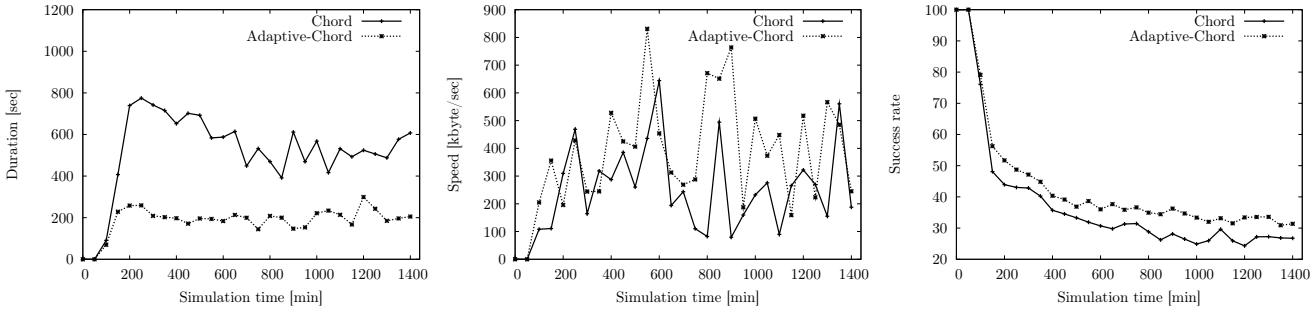
(a) Number of nodes in simulations with churn (b) Scenario without churn (c) Scenario with churn

Fig. 6. Average Number of Overloaded Peers in the Scenarios busy_unstable_*



(a) Average download duration (b) Average download speed (c) Average success ratio

Fig. 7. Simulation Results for Scenario busy_unstable_nochurn



(a) Average download duration (b) Average download speed (c) Average success ratio

Fig. 8. Simulation Results for Scenario busy_unstable_churn

when peers go offline. Nevertheless, also in this case, Adaptive Chord is serving more downloads than Chord can. All these improvements can be seen as a result of the goal to lower the number of overloaded peers. Figure 9 shows the success rate of retrieving files that are actually available in the network. The average success rate in Adaptive Chord after ten hours simulated time is 88% while Chord's success rate is at 66%. Thus, even in a churn scenario the quality of file availability is significantly raised by applying our proposed method.

5) Maintenance Overhead: Since using a second overlay introduces additional communication overhead used to sustain the structure of the overlay, we measured the traffic that occurs due to the various management messages of the overlays. Figure 10 depicts the amount of traffic each peer spends at sending control messages used to sustain the structure of the corresponding DHT in a churn scenario. It can be seen that the

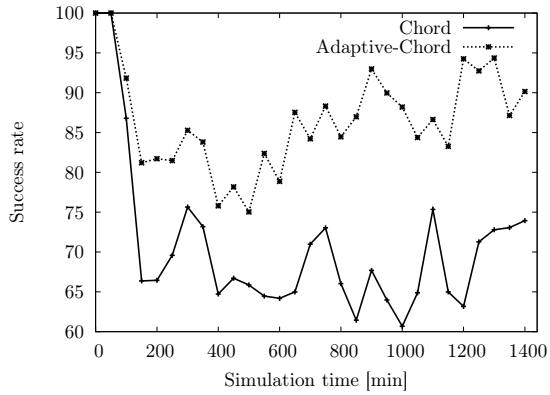


Fig. 9. Success Ratio in Relation to the Number of Available Files.

control traffic in Adaptive Chord is about 6 times higher than in Chord, still both are below one tenth of a kilobyte. Thus, the difference is not considered as significant; it has no impact

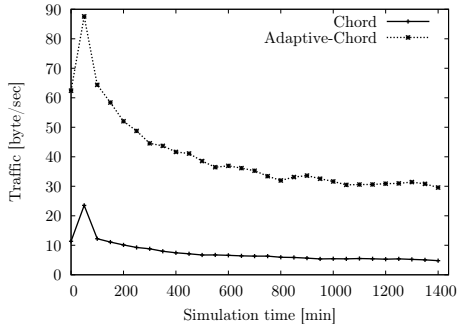


Fig. 10. Average Control Traffic per Peer in a Churn Scenario

on the performance of the networks.

6) Summary: The evaluation has shown that the concept of Adaptive Chord reduces the number of overloaded peers in the network significantly and thus leads to faster download speed and shorter download times. With sufficient nodes available in the capacity overlay, fitting file suppliers can be found and the download success, speed and time of file serving improves.

V. Conclusions

We addressed the issue of overloaded nodes in distributed hash tables that cannot serve all queries for data objects they host. As a result, file transfers are slow and fail in a rather large fraction of cases. In order to address this issue, we follow the idea of storage indirection and present the concept of a capacity overlay for capacity-based storage node selection. In our variant, Adaptive Chord, overloaded peers shift the duty of storing load critical data objects to powerful supplier peers. For that, Adaptive Chord maintains beside the traditional data overlay a second Chord overlay for querying peer capacities. Evaluation shows the presented concept is significantly decreasing the number of overloaded nodes in comparison to traditional Chord. With the much lower number of overloaded nodes in the network, the average download speed in Adaptive Chord raises up to 232% in comparison to Chord, download time drops to 10% and a (much) larger fraction of queries is answered successfully even under churn. The concepts presented in this paper are applicable not only to Chord but to other DHTs as well.

Since our proposed method can be seen as an easy-to-use add-on to existing applications built on top of DHTs, the insights given in this work can be used to create a significant impact not only on the robustness of those applications but also on their usability. As content stored in the DHT is delivered much faster, the applications can be made more interactive, leading to a higher attractiveness to their users.

References

[1] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: Feasibility of Transitive Cooperation for Privacy on a Decentralized Social Network," in *Proc. of IEEE WOWMOM '09*, 2009.

[2] S. Buchegger *et al.*, "PeerSoN: P2P Social Networking - Early Experiences and Insights," in *Proc. of ACM SNS '09*. ACM, 2009.

[3] K. Graffi *et al.*, "LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks," in *Proc. of IEEE CCNC '11*, 2011.

[4] K. Graffi *et al.*, "A Distributed Platform for Multimedia Communities," in *Proc. of IEEE ISM '08*, 2008.

[5] L. Breslau *et al.*, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proc. of INFOCOM '99*, 1999.

[6] F. Benevenuto *et al.*, "Characterizing User Behavior in Online Social Networks," in *Proc. of ACM SIGCOMM IMC '09*, 2009.

[7] OECD, *Broadband Growth and Policies in OECD Countries*. Organisation for Economic, Co-Operation and Development, 2008.

[8] R. Süselbeck *et al.*, "Efficient Bandwidth Estimation for Peer-to-Peer Systems," in *Proc. of IEEE P2P '11*, 2011.

[9] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," in *Proc. of IPTPS '04*, 2004.

[10] I. Stoica *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of ACM SIGCOMM '01*, 2001.

[11] E. Harjula and M. Ylianttila, "Load Balancing Models for DHT-based Peer-to-Peer Networks," IETF, Internet Draft <draft-harjula-p2psip-loadbalancing-survey-01.txt>, 2010.

[12] F. Hong *et al.*, "VChord: Constructing Peer-to-Peer Overlay Network by Exploiting Heterogeneity," in *Proc. of the EUC '04*, 2004.

[13] J. Ledlie and M. I. Seltzer, "Distributed, Secure Load Balancing with Skew, Heterogeneity and Churn," in *Proc. of IEEE INFOCOM '05*, 2005.

[14] S. Surana *et al.*, "Load Balancing in Dynamic Structured Peer-to-Peer Systems," *Perform. Eval.*, vol. 63, no. 3, 2006.

[15] M. Abdallah and E. Buyukkaya, "Fair Load Balancing under Skewed Popularity Patterns in Heterogeneous DHT-based P2P Systems," in *Proc. of IASTED PDCS'07*, 2007.

[16] M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide, "Dynamic Load Balancing in Distributed Hash Tables," in *Proc. of IPTPS '05*, 2005.

[17] Z. Zhang *et al.*, "Resilient capacity-aware multicast based on overlay networks," in *Proc. of IEEE ICDCS '05*, 2005.

[18] J. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," in *Proc. of IPTPS '03*, 2003.

[19] A. I. T. Rowstron and P. Druschel, "Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility," in *Proc. of ACM SOSP '01*, 2001.

[20] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays," in *Proc. of USENIX NSDI '04*, 2004.

[21] M. Srivatsa, B. Gedik, and L. Liu, "Scaling Unstructured Peer-to-Peer Networks With Multi-Tier Capacity-Aware Overlay Topologies," in *Proc. of IEEE ICPADS '04*, 2004.

[22] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, 2007.

[23] F. Dabek *et al.*, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proc. of IPTPS '03*, 2003.

[24] S. Bianchi *et al.*, "Adaptive Load Balancing for DHT Lookups," in *Proc. of IEEE ICCCN '06*, 2006.

[25] O. Beaumont, L. Eyraud-Dubois, and Y. Won, Joon, "Using the Last-mile Model as a Distributed Scheme for Available Bandwidth Prediction," in *EuroPar '11*, 2011.

[26] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," in *Proc. of IEEE FOCS '03*, 2003.

[27] K. Graffi *et al.*, "SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems," in *Proc. of IEEE ICPADS '08*, 2008.

[28] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator Experiences and Lessons Learned," in *Proc. of IEEE P2P '11*, 2011.

[29] A. Kovacevic, S. Kaune *et al.*, "Benchmarking Platform for Peer-to-Peer Systems," *it - Information Technology*, vol. 49, no. 5, 2007.

[30] D. Stingl *et al.*, "PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems," in *Proc of HPCS '11*, 2011.

[31] OECD, "OECD Bandwidth Report," 2007. [Online]. Available: http://www.oecd.org/sti/ict/broadband

[32] 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, pp. 61–61, 2002.

[33] I. Stoica *et al.*, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions of Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[34] 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.