Delay Analysis of a Chord-based Peer-to-Peer File-Sharing System

Andreas Binzenhöfer and Phuoc Tran-Gia
Department of Distributed Systems
Institute of Computer Science
University of Würzburg, Am Hubland, 97074 Würzburg, Germany
{binzenhoefer, trangia}@informatik.uni-wuerzburg.de

Abstract—In recent years emerging file sharing systems like Gnutella, eDonkey, Overnet, and Kazaa strongly influenced the behaviour of Internet traffic. These platforms employ different peer-to-peer mechanisms, where the application areas are just beginning to shift from undemanding content sharing towards new business case services. Those new requirements brought out new peer-to-peer overlay architectures like Chord and Kademlia based on Distributed Hash Tables. The new algorithms satisfy the needs of distributed applications like, e.g., telephone directories supporting “anywhere” VoIP. In this paper we investigate the delay of the search process in such a peer-to-peer directory service, where the Chord algorithm is used and the peers are connected through the internet with varying end-to-end transport delay. To guaranty real-time services, quantiles of the search delay are analytically computed. The study also contains the analysis of the scalability of the system, where the impact of the peer population on the search delay characteristic is investigated.

Index Terms—P2P, Scalability, Chord, DHT, Performance, Delay Analysis

I. INTRODUCTION

In the last decade most Internet applications relied on the client-server architecture. However, a centralized single point of failure did not prove robust and resilient enough to cope with the growing demands of Internet users. As a consequence thereof, distributed systems, that do not rely on a central control entity came into existence. In a distributed peer-to-peer (P2P) network each peer is considered equal. That is, each peer runs the same piece of software and performs exactly the same tasks as any other peer in the network.

In order to build large networks like national or global telephone directories, the underlying mechanisms have to be scalable. Therefore, scalability of P2P applications became an independent field of research. The first P2P networks either relied on a central server, did not scale to a large number of nodes [12] or had the population of peers divided into different groups of unequal responsibilities like, e.g., the so called superpeers in Kazaa [7]. Recently, however, the most promising approaches started using distributed hash tables (DHTs) to organize their P2P overlay networks.

This work was partially founded by the European Commission within the framework of the EuroNGI project.

Independent of the type of application, one of the main tasks of a stand-alone P2P network is the retrieval of data location. So far, the time needed to complete a search in regular file-sharing systems was not really critical to the end-user since file download time exceeded the preceding lookup time of the files location by magnitudes. Real-time applications with certain quality of service demands, like VoIP telephony, chatting, or instant messaging on the other hand are dependent on the time needed to find their communication partner. The arising problem in terms of scalability is that in a large population of peers, not every peer is able to know about the location of every other peer. Moreover, virtual neighborhood relationship in the overlay network usually does not correspond to physical proximity as well.

DHT based P2P algorithms like Chord [1] only need to store the location of \(O(\log_2(n))\) other peers where \(n\) is the number of peers in the overlay network. They are furthermore able to retrieve information stored in the distributed network by using \(O(\log_2(n))\) messages to other peers. This statement, however, is very vague, since it only tells us the order of magnitude of the search delay and does not provide us with sufficient details on search duration statistics. As a matter of fact the physical link delay strongly influences the performance of searches in a P2P overlay network. The physical network behavior, however, is highly probabilistic. In this paper the impact of network delay variation on search times in DHT based P2P systems is evaluated in more detail. The main goal is to prove scalability in very large Chord rings, to be able to guaranty certain quality of service demands in large peer populations. The enormous complexity of such systems makes an evaluation by simulation on packet level rather intractable. We therefore deduce an analytical performance model for real-time applications based on the Chord algorithm. While the calculation of the mean of the search duration is quite straightforward, the computation of the quantiles of the search duration is more complex. The quantiles, however, have an important impact on the quality of service experienced by the end user. Making some plausible assumptions, we therefore calculate quantiles for the maximum search duration.

The paper is structured as follows. Section II describes the concept of DHTs in a Chord ring using a simple example. The assumptions made in this paper are summarized and explained...
in detail. In Section III the network model is introduced, the
distribution of the number of hops used in a search process and
the distribution of the search time are calculated accordingly.
A simple example as well as parametric studies considering
the search time variation are presented in Section IV. Section
V finally summarizes and concludes the paper.

II. DISTRIBUTED HASH TABLES USING CHORD RING
STRUCTURES

Distributed hash tables (DHTs) represent a decentralized
mechanism for associating different kinds of objects and peers
with hash values. The most commonly used hash functions are
MD5 [14] and SHA1 [13]. A DHT is mainly used to distribute
peers and objects as uniformly as possible to the hash functions
codomain, i.e. the identifier space. The hash function maps
each peer (e.g. by using its IP address) and each object (e.g.
by using the file itself) to a value in the identifier space. As can
be seen in Figure 1, a value is assigned to each peer and object,
in such a way that peers and objects intermix in the identifier
space. Each participating peer is then responsible for all those
objects, whose hash values lie between the peers own hash
value and the hash value of the peer immediately preceding
the peer. In Figure 1 object $z$, e.g. is hashed between peer $x$
and its preceding peer $y$. Information about object $z$ is thus
stored at peer $x$. To cope with the boundaries of the identifier
space the identifier space is transformed into a circle in such
a way that the hash values are ascending clockwise in the
evolving ring. This ring is called the Chord ring. Information
about a document is thus stored at the first peer, whose hash
value succeeds the document’s hash value on the Chord ring.
For routing purposes a peer knows its immediate successor on
the ring. If a peer searches for a document, it will forward the
query to its successor, which in turn will forward the query to
its own successor until the search hits the peer responsible for
the searched document. Once the responsible peer is found, it
will transmit the answer directly to the originator, i.e. the peer
seeking the information. As a peer needs $O(n)$ messages to
complete this kind of search, it also maintains a finger table,
i.e. a list of peers called fingers. These fingers are used as
shortcuts through the ring to speed up the search process. As
shown in [1] the finger table of a peer $z$ consists of $O(\log_2(n))$
distinct entries whereby the $i$-th entry in a peers finger table
contains the identity of the first peer that succeeds $z$’s own
hash-value by at least $2^{i-1}$ on the Chord ring. That is, peer $z$
with hash value $id_z$ has its fingers at $id_z + 2^{i-1}$ for $i = 1$ to
$\log_2(n)$. Figure 2 illustrates a simple example using a Chord-
answer the search locally, it forwards the query accordingly. Otherwise, the search is finished and the finger directly returns the answer back to the searching peer \( z \). Figure 2(b) shows a query of peer 1 looking up peer 12. The search is forwarded to peer 9, as peer 9 is peer 1’s finger most immediately preceding peer 12. Since peer 9 is not able to answer the search locally it recursively forwards the query to its finger at peer 11. Again peer 11 is not able to answer the search locally and forwards the query to peer 12, which is finally able to send an answer back to the originator peer 1. This way queries can be answered using \( O(\log_2(n)) \) messages. This kind of search is called recursive, since each peer participating in the search forwards the query recursively to its closest finger. As opposed to iterative queries, where each peer involved in the query reports back to the originator, recursive searches only take about 0.6 times as long according to [11]. We therefore concentrate on recursive lookups for the remainder of this paper.

Since an evaluation by simulation on packet level is very time-consuming for very large populations, we prove the scalability of search queries in oversized Chord rings analytically. In the next section we therefore deduce an analytical performance model making the following two assumptions:

- For the sake of simplicity we assume that each finger entry of a peer always directly hits another peer. That is, a peer \( z \) with hash value \( id_z \) has its fingers directly at \( id_z + 2^{i-1} \) for \( i = 1 \) to \( \log_2(n) \).
- We assume a perfect hash function. That is, all peers and documents are distributed uniformly in the identifier space. Furthermore, each document is looked up with the same probability independent of its location on the ring. In other words, each peer will be responsible for the same number of documents and therefore each peer will answer the same number of queries.

### III. Delay Analysis

In this section the delay distribution function, i.e. the time needed to complete a search in a Chord ring, will be analyzed. Since the physical path delay strongly influences the performance of searches in DHT based P2P systems, the impact of network delay variation is taken into consideration as well. We use the following random variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_N )</td>
<td>describes the delay of a query packet, which is transferred from one peer to a successor peer</td>
</tr>
<tr>
<td>( T_A )</td>
<td>represents the time needed to transmit the answer from the peer (having the answer) back to the originator</td>
</tr>
<tr>
<td>( T )</td>
<td>describes the total search duration</td>
</tr>
<tr>
<td>( X )</td>
<td>indicates how many times a query has to be forwarded until it reaches the peer having the answer. ( X ) will be denoted as the peer distance</td>
</tr>
<tr>
<td>( H )</td>
<td>number of overlay hops needed to complete a search, i.e. the number of forwards of the query plus one hop for the transmission of the answer</td>
</tr>
<tr>
<td>( n )</td>
<td>size of the Chord-ring</td>
</tr>
</tbody>
</table>

We distinguish between \( T_N \) and \( T_A \), as the size (and therefore the delay) of a search packet and an answer packet may be unequal. The answer might, e.g. consist of multiple packets containing a detailed reply to the query.

#### A. Computation of Peer Distance Distribution

In this section we compute the probability distribution of the peer distance \( X \). It is needed later on to analyze the distribution of the search duration. On basis of the assumptions made in Section II each peer is looked up with the same probability. Since a search is recursively forwarded to the closest finger, we are able to calculate the number of hops needed to reach the peer, that answers a specific query. We derive the probability \( p_i = P(X = i) \) that the searched peer is exactly \( i \) hops away from the searching peer. The next section deals with the special case of peer populations of binary exponential size.

1) Special Case of Binary Exponential Peer Populations:

To provide an overview, we start with Chord rings whose size is a power of 2. In an overlay network of this specific size \( n = 2^k \), \( k = \log_2(n) \) is an integer and each peer has \( k \) distinct fingers. The assumptions made in Section II provide that the \( i \)-th finger of a peer \( z \) is used for searches for all peers whose corresponding hash values fall between \([id_z + 2^{i-1}, id_z + 2^i - 1]\) where \( id_z \) is the hash value of peer \( z \). Again this can be illustrated using the example in Figure 2(a). In this context the \( 4^{th} \) finger of peer 1, which is pointing to peer 9, is responsible for all peers between \([9, 16] = [1 + 2^{4-1}, 1 + 2^4 - 1]\).

Taking this into account, we construct Table I consisting of four columns. The first column represents the peer distance \( X \). The second column states the number of hops \( H \) needed to complete a search. In the case of \( X = 0 \) the searched document lies in the same peer. A search answered locally likewise requires 0 hops. To complete a search answered by a peer that is \( X > 0 \) hops away, however, we need \( X \) hops to reach that peer and one additional hop to send the answer back to the originator. Altogether \( X + 1 \) hops are needed to perform this kind of search. Column 4 finally describes the random variable \( T \) representing the time needed to complete such searches by adding \( X \) times the delay of a forwarded query packet plus the time needed to transmit the answer back to the originator. The probability \( p_i = P(X = i) \) in column 3 is governed by the following theorem:

**Theorem:** The probability that the searched peer is exactly \( i \) hops away from the searching peer in a Chord ring of size \( 2^k \) (and thus with \( \log_2(2^k) = k \) fingers) with symmetric search space and uniformly distributed keys is

\[
p_i = P(X = i) = \frac{k}{2^k}
\]

The proof can be found in the Appendix. The assumptions regarding the symmetric search space and the uniformly distributed keys assure that each peer will be looked up with the same probability.
TABLE I
Peer distance distribution and search time

<table>
<thead>
<tr>
<th>X</th>
<th>H</th>
<th>( P(X = i) )</th>
<th>search time ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( p_0 = \frac{\log_2(n)}{n} )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>( p_1 = \frac{\log_2(n)}{n^2} )</td>
<td>( T_A + T_N )</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>( p_2 = \frac{\log_2(n)}{n^3} )</td>
<td>( T_A + T_N + T_N )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( i )</td>
<td>( i+1 )</td>
<td>( p_i = \frac{\log_2(n)}{n^{i+1}} )</td>
<td>( T_A + \sum_{x=1}^{i} T_N )</td>
</tr>
</tbody>
</table>

2) Arbitrary Number of Peers: So far we considered the special case of a binary exponential peer population. We now extend the model to an arbitrary number of peers and keep the assumptions made in Section II. In a Chord ring of arbitrary size \( n \) we have \( k = \lceil \log_2(n) \rceil \) distinct fingers. That is, a Chord ring of this size maintains just as many different fingers as a Chord ring of size \( m = 2^k \), the next largest power of 2. Since we are assuming serially numbered peers, the \( i \)-th finger of a peer \( z \) still points to the same peer \( \text{id}_z + 2^{k-1} \) for \( i = 1 \) to \( \lceil \log_2(n) \rceil \). In other words we can compare Chord rings of arbitrary size \( n \) to Chord rings of binary exponential size \( m \), except that \( m - n \) peers are missing between the last finger and the searching peer itself.

![Fig. 3. Finger-table of peer 1 in a 12 peer Chord ring](image)

Figure 3 illustrates this issue for a Chord ring of size 12. The figure resembles Figure 2(a) insofar as searches for peers 1 to 8 originating at peer 1 still require the same number of hops. The only difference is that the last finger pointing to peer 9 is now covering less peers and is thus responsible for less searches. On account of this, we divide a Chord ring of arbitrary size into two parts to calculate the peer distance distribution \( X \). The first part consisting of the first \( 2^{k-1} \) peers, the second part including the remaining peers.

In conjunction with the preceding theorem we conclude the following corollary, calculating the number \( f_n(i) \) of peers in a Chord ring of arbitrary size \( n \) that are \( i \) hops away from the searching peer \( z \):

**Corollary:** The number \( f_n(i) \) of peers in a Chord ring of arbitrary size \( n \) (and therefore with \( \lceil \log_2(n) \rceil \) distinct fingers) that are \( i \) hops away from the searching peer is:

\[
f_n(i) = \begin{cases} 
\binom{k}{i}, & \text{if } n = 2^k \\
\binom{k-1}{i} + f_n(2^{k-1} - (i - 1)), & \text{if } 2^{k-1} < n < 2^k 
\end{cases}
\]

![Fig. 4. Phase diagram of the search duration \( T \)](image)

The corollary exploits the fact that there are no changes in the structure of the first \( 2^{k-1} \) peers compared to an independent Chord ring of size \( 2^{k-1} \) and recursively calculates the hops needed for searches covered by the last finger. Note that in the recursive calculation we have to subtract one hop needed to reach the responsible finger. Finally to get \( p_i \) in the arbitrary case, \( f_n(i) \) will be divided by \( n \), i.e.

\[
p_i = \frac{f_n(i)}{n}
\]

B. Search Delay Analysis

As a result of the last section we now know the peer distance distribution \( X \). From this we derive the length in hops of the path a particular search-query takes through the network. We also know the probability \( p_i \) that a search takes exactly this path. Using these basic relations we can compute the distribution of the search delay as a function of the network delay characteristics. First, the basic relations in our model are illustrated followed by the direct computation of the mean and the variation of the search duration. Section IV presents some parameter studies to exemplify how the coefficient of variation of the network transmission delay influences the search duration.
The phase diagram of the search delay is depicted in Figure 4. A particular path $i$ is chosen with probability $p_i$ where phase $i$ consists of $i$ network transmissions $T_N$ to forward the query to the closest known finger and one network transmission $T_A$ to send the answer back to the searching peer. By means of the phase diagram, the generating function and the Laplace-Transform respectively can be derived to cope with the case of discrete-time or continuous-time network transfer delay.

The mean and the coefficient of variation of the search delay are such:

$$E[T] = \sum_{i=1}^{k} p_i \cdot E[T|k = i]$$

$$= \sum_{i=1}^{k} p_i \cdot (E[T_A] + i \cdot (E[T_N]))$$

$$E[T^2] = \sum_{i=1}^{k} p_i \cdot E[T^2|k = i]$$

$$= \sum_{i=1}^{k} p_i \cdot (VAR[T_A] + i \cdot VAR[T_N]$$

$$+ (E[T_A] + i \cdot E[T_N])^2)$$

and

$$c_T^2 = \frac{E[T^2] - E[T]^2}{E[T]^2}$$

IV. NUMERICAL RESULTS

In this chapter we present numerical results to illustrate the dependency of the search duration on the variation of the network transfer delay and to give insight into the scalability of the Chord-based file sharing mechanism. First we will show the mean and the coefficient of variation of the search duration. Subsequently, the shape of the search delay distribution function will be discussed, followed by the quantile analysis, i.e. the guaranty that $\alpha$ percent of searches will need less than $t$ seconds.

Regarding the results in this section, the delay $T_N$ is assumed to be identical to the delay $T_A$. To unify the following parametric study the delay $T_N$ is further modeled by means of a two-parameter negative-binomially distributed random variable. If not stated otherwise, the coefficient of variation $c_{T_N}$ of $T_N$ is set to 1 and the mean $E[T_N]$ of $T_N$ is set to 50ms, since 50ms is the value assumed in the original Chord Paper [1]. Furthermore, we divide the obtained results by $E[T_N]$ where appropriate to obtain more general conclusions.

Figure 5 shows the mean search delay as a function of the size of the Chord ring. We can observe that the search delay rapidly increases at smaller values of $n$, but stays moderate for very large peer populations. The curve is not strictly monotonically increasing as expected since a small decrease can be seen when the population $n$ just exceeds a binary exponential value $2^i$. This effect can be explained as follows:

Once the size of the population crosses the next power of 2, the finger table of each peer grows by one entry. Thus, the mean search duration slightly decreases at this point.

The coefficient of variation $c_T$ of the search delay $T$ is depicted in Fig. 6 as a function of the peer population, for different transmission delay coefficients of variation. The variation of the search duration increases with $c_{T_N}$. However, it decreases as the Chord size increases, due to the increasing number of hops needed in larger Chord populations.

This effect is also illustrated in Fig. 7, where the dependency of $c_T$ on $c_{T_N}$ is analyzed. Again it can be seen that $c_T$ is smaller than $c_{T_N}$. The size of the Chord population itself has a comparatively small effect on $c_{T_N}$.

In Figures 8 and 9 we study the dependence of the entire distribution function of the search delay on the network latency.
variation $c_{T_N}$ and the peer population $n$, respectively. The size of the peer population in Figure 8 is set to $10^5$ peers. As expected, the probability that a search takes longer increases together with the coefficient of variation of the network latency $c_{T_N}$. The curves in Figure 8 intersect as they share the same mean $E[T_N]$ but have different coefficients of variation $c_{T_N}$.

Figure 9 proves the scalability of the search delay. By increasing the size of the Chord ring from $10^3$ peers to $10^6$ peers the search delay distribution does not escalate exponentially but increases by a linear factor. The chosen values of $n$ correspond approximately to current file sharing networks like the edonkey network.

Figures 10 and 11 depict the quantile of the search delay $T$. In Figure 10 different quantiles for the search delay are taken as a parameter. For example the curve with the 99%-quantile indicates that 99 percent of search durations lie below that curve. For a peer population of, e.g., $n=3000$ in 99 percent of all cases the search delay is less then roughly 15 times the average network latency. It can be seen that the curves indicate bounds of the search delay, which can be used for dimensioning purposes, e.g. to know the quality of service in a search process with real-time constraints like looking at a phone directory, taking into account the patience of the users. Compared to the mean of the search delay the quantiles of the search delay are on a significantly higher level. Still the search delay scales in an analogous manner for the search delay quantiles.

Figure 11 depicts the 99%-quantile of the search delay, again with the coefficient of variation of $T_N$ as a parameter.
APPENDIX

**Theorem:** The probability that the searched peer is exactly \( i \) hops away from the searching peer in a Chord ring of size \( 2^k \) (and thus with \( \log_2(2^k) = k \) fingers) with symmetric search space and uniformly distributed keys is

\[
p_i = P(X = i) = \binom{k}{i} \frac{1}{2^k}
\]

**Proof:** We argue by induction.

**Basis:**
For \( k = 0 \), in a ring with \( 2^0 = 1 \) node, there is exactly \( \binom{0}{0} = 1 \) node, that is 0 hops away from the only peer \( z \). Therefore \( p_0 = \frac{\binom{k}{0}}{2^k} \).

For \( k = 1 \), in a ring with \( 2^1 = 2 \) nodes, there is exactly \( \binom{1}{0} = 1 \) node, that is 0 hops away from peer \( z \) and exactly \( \binom{1}{1} = 1 \) node, that is 1 hop away from peer \( z \). Therefore \( p_0 = \frac{\binom{1}{0}}{2^k} \) and \( p_1 = \frac{\binom{1}{1}}{2^k} \).

**Induction hypothesis:**
Assume the theorem is true for \( y \leq k \).

**Induction step:**
Prove the theorem is also true for \( k + 1 \).

To calculate the number of peers that are \( i \) hops away from a peer \( z \) in a chord ring of size \( 2^{k+1} \), we divide the chord ring into two parts consisting of the first \( 2^k \) and the last \( 2^k \) peers respectively. We then calculate the number of peers that are \( i \) hops away from peer \( z \) in those two parts of the original ring and simply add those two numbers up.

First \( 2^k \) nodes: In a chord ring of size \( 2^{k+1} \) a peer \( z \) has \( k + 1 \) fingers. The first \( k \) fingers are responsible for the first \( 2^k \) peers. By induction hypothesis there are exactly \( \binom{k}{i} \) peers that are \( i \) hops away from peer \( z \) in this part of the ring.

Last \( 2^k \) nodes: The \((k+1)\)-th finger covers the remaining \( 2^k \) peers in the original chord ring. By induction hypothesis there are exactly \( \binom{k}{i} \) peers that are \( i \) hops away from the \((k+1)\)-th finger in this part of the ring. Since the \((k+1)\)-th finger is exactly 1 hop away from peer \( z \), there are \( \binom{k}{i-1} \) peers in this part of the ring that are \( i \) hops away from peer \( z \) (one hop to reach the finger-peer and \( i - 1 \) hops to reach the corresponding peer).

Altogether there are \( \binom{k}{i} + \binom{k}{i-1} = \binom{k+1}{i} \) peers that are exactly \( i \) hops away from peer \( z \). Since there are \( 2^{k+1} \) peers the probability that another peer is exactly \( i \) hops away from peer \( z \) is \( p_i = \frac{\binom{k+1}{i}}{2^{k+1}} \).

**Conclusion:** Together, the basis and the induction step imply that the theorem holds for all possible cases, i.e., in a chord ring of size \( n = 2^k \) the probability that the searched peer is
exactly $i$ hops away from the searching peer $z$ is

$$p_i = \frac{k}{2^k}$$

Note that the proof of this theorem as well as the corollary in Section III are based on serially numbered peer ids. That is, in a chord ring of size $n$ the peer ids are numbered $1, 2, \ldots, n$, such that peer number $z$ has $id_z = z$. In a real chord ring of size $n$, identifier space size $N = 2^j$ and equally distributed peers, however, peer number $z$ has

$$id_z = 1 + (z - 1) \cdot \left\lceil \frac{N}{n} \right\rceil$$

In this case the assumption that the fingers of peer number 1 are directly pointing to peers number

$$1 + 2^{i-1}, \; i = 1, \ldots, k$$

is no longer obvious. Instead, according to the Chord algorithm the fingers of peer number 1 are pointing to the first peers, whose ids are directly succeeding $1 + 2^{j-b}, \; b = 1, \ldots, j$ on the Chord ring respectively. That is, the fingers are pointing to peers number $z_b$, where

$$z_b = \min\{x : 1 + (x - 1) \cdot \left\lceil \frac{N}{n} \right\rceil \geq 1 + 2^{j-b}, \; b = 1, \ldots, j\}$$

In the special case of binary exponential peer populations $n = 2^k$ it follows that

$$z_b = \min\{x : 1 + (x - 1) \cdot \frac{2^j}{2^k} \geq 1 + 2^{j-b}, \; b = 1, \ldots, k\}$$

Since $x$ has to be an integer the fingers of peer 1 are thus pointing to peers number $1 + 2^{k-b}, \; b = 1, \ldots, k$, which are exactly the same peers as in equation 1 where the peer ids were numbered serially. In the special case of $n = 2^k$ the proof can thus be extended to non-contiguous peer ids. Whether the above also holds in the arbitrary case is subject of further study.

REFERENCES