Chord++: Low-Congestion Routing in Chord

Baruch Awerbuch
Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
baruch@cs.jhu.edu

Christian Scheideler
Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
scheideler@cs.jhu.edu

January 20, 2004

Abstract

Efficiently determining the node that stores a data item in a distributed network is an important and challenging problem. In an influential paper [8], Stoica, Morris, Karger, Kaashoek and Balakrishnan solved this problem in an elegant way by suggesting Chord, a peer-to-peer system based on a combination of a hypercubic network with an indexing method, called consistent hashing, that allows efficient lookup. Theoretical analyses have shown that Chord is incrementally scalable, with insertion and lookup costs scaling logarithmically with the number of Chord nodes. The algorithm for the lookup operation is (like the algorithms for inserting and deleting a peer) extremely simple, and it has been shown that every lookup request only has to traverse a logarithmic number of edges in Chord to locate the node storing the requested item. However, so far no one has been able to prove reasonably low bounds on the congestion caused by the lookup algorithm when executing several requests in parallel; the way nodes are connected in Chord makes this extremely difficult to analyze. To get around this problem, we modify Chord by giving each user several nodes and we suggest a new lookup algorithm that corrects the path of the original lookup algorithm once in a while. This algorithm can be analyzed, and we show that any permutation routing problem as well as random routing problems can be handled by it with a logarithmic congestion and dilation with high probability. Hence, by upgrading Chord in a suitable way, which we call Chord++, concurrent lookup requests can be handled in a highly efficient manner. This also implies efficient algorithms for concurrent delete, insert, and join requests.

As a byproduct of our results we show that when using our routing algorithm in the original Chord network, $O(\log^2 n)$ congestion can be achieved, where n is the number of nodes. Furthermore, it follows from our congestion bounds that Chord has a node expansion of $\Omega(1/\log n)$, which matches the best expansion results known so far for other peer-to-peer systems.

1 Introduction

In order to manage data in a peer-to-peer system, five operations have to be supported:

- u.Join(v): User u wants to join the system by contacting user v that is already a part of it.
- u.Leave(): User u wants to leave the system.
- u.Insert(key): u wants to insert a data item with name key into the system.
- u.Delete(key): u wants to delete the data item with name key from the system.
- u.Lookup(key): u wants to find the data item with name key in the system.

In an influential paper [8], Stoica, Morris, Karger, Kaashoek and Balakrishnan came up with an elegant solution for managing data in a peer-to-peer system, called Chord. Chord is basically a combination of a hypercubic network with an indexing method called consistent hashing [3]. Consistent hashing works as follows:

Let V be the set of all possible user names and D be the set of all possible names for data items. Consider the two hash functions $h : V \to [0, 1)$ and $g : D \to [0, 1)$ that map each user or data item to a real number in $[0, 1)$. Then the consistency condition that has to be maintained at all times is that each data item $x \in D$ currently in the system must be stored at the user $v \in V$ with minimum $h(v)$ so that $h(v) \geq g(x)$. If h and g are random or pseudo-random, it turns out that keeping the placement consistent is very cheap: First of all, random functions h and g ensure that at any time, the expected number of data items stored in a node is the same for all nodes. Hence, if the current number of nodes in the system is n, then the expected amount of data replacements when including or excluding a node is roughly a $1/n$ fraction of the total amount of data in the system.

Hence, under the assumption that the hash functions used have (pseudo-)random properties, it is known that Chord achieves logarithmic bounds on the average degree and average update cost, and every lookup request only has to traverse a logarithmic number of edges in the Chord network to find the requested data item. However, so far no one has been able to analyze the congestion caused by routing lookup requests in Chord. Bounding the congestion for concurrent lookup requests is important, because in a peer-to-peer system many users may issue lookup requests concurrently, and a lookup mechanism with a high congestion would certainly severely limit the usefulness of the peer-to-peer system.

It seems intuitively obvious that the routing mechanism suggested for Chord in [8] should be able to route messages concurrently without congesting. In fact, we conjecture that this is indeed the case, but it seems to be extremely hard to prove. Instead we suggest a new lookup method that can be analyzed. One of the basic mechanisms we employ is a fundamental paradigm introduced by Valiant and Brebner [9]: go to a random destination first, and then to the final destination. However, to make the proof go through, we still also need to make some local corrections in the packet trajectory. The final algorithm is still very simple, and is very similar to the original mechanism in Chord [8]. This is nevertheless the first proof that concurrent lookup requests can be served with low congestion in this important distributed topology, namely $O(\log n)$ with high probability, where n is the number of users in Chord. The techniques used in the algorithm and the proof are general enough so that it should be possible to come up with a low-congestion lookup strategy also for other peer-to-peer systems based on consistent hashing, such as CAN [6], Pastry [7], and Viceroy [4]. Certainly, if lookup operations can be done concurrently with low congestion, then also update operations (i.e. inserting or deleting data items) and join requests can also be handled concurrently with low congestion. Hence, supporting lookup operations efficiently is of central importance for a peer-to-peer network to be useful in practice.

Another important consequence of being able to support lookup requests efficiently is that the network must have a high expansion. In particular, a consequence of our bounds is that Chord has expansion of $\Omega(1/\log n)$ with high probability.

1.1 Related work

Our results are the first to prove $O(\log n)$ congestion bounds for routing in a peer-to-peer system, without any simplifying assumptions. Previous work achieves $O(\log^2 n)$ bounds under such circumstances. Specifically:

- The butterfly network of Viceroy [4] achieves congestion $O(\log^2 n)$ with high probability [4].
- Skip Graphs [1] have an expansion of $\Omega(1/\log n)$ [1], but no direct bound on congestion has been shown. It appears that $O(\log^2 n)$ bound on routing congestion can be proved.
- DeBruijn-graph based peer-to-peer systems were recently, and independently from our work, analyzed by Naor and Wieder [5]. This work presents a congestion bound of $O(\log n)$ with high probability under a simplifying assumption, namely assuming a *smooth* distribution of node names. This smoothness assumption is usually *not* fulfilled for most of the existing peer-to-peer systems with (pseudo-)random names [8, 7, 10] (i.e. when using hash functions for naming). Getting rid of the smoothness assumption appears to require an extra $\Theta(\log n)$ factor, yielding an overall bound of $O(\log^2 n)$.

1.2 Notation

Expansion and searchability, and results including how to go from permutations to path systems to routing arbitrary routing problems.

2 Chord

In this section we start with the original Chord overlay network and present and analyze the performance of routing in Chord.

2.1 The Chord overlay network

The Chord overlay network is defined as follows. Suppose that we have a system currently consisting a set V of n peers (also called nodes in the following), and further suppose we have a hash function $h : V \to [0, 1)$ with all properties of a completely random function that maps nodes to real values in the interval $[0, 1)$. Then the basic structure of Chord is formed by a directed cycle, the so-called *Chord ring*, in which all nodes are ordered according to their hash values. In addition to this, every node v has edges to nodes $p_i(v)$, called *fingers* with

$$p_i(v) = \mathsf{argmin}\{w \in V \mid h(w) \geq h(v) + 1/2^i\}$$

for every $i \geq 1$. Also, for fault tolerance reasons, it has been suggested that every node v have a list of $s \log n$ *successor pointers*, i.e. edges to its nearest $s \log n$ successors in the Chord ring, where s is a sufficiently large constant.

2.2 Routing in Chord

In order to route a message from node u to node w, the Chord overlay network uses a path $p(u, v)$ consisting of a sequence of nodes $v_0, v_1, v_2, \ldots, v_\ell$ with the property that $v_0 = u$, for all $j \in \{0, \ldots, \ell - 1\}$, $v_{j+1} = p_{i_j}(v_j)$ where i_j is the smallest integer so that $h(v_{j+1}) \leq h(w)$, and $v_{\ell-1}$ is the first node that has a successor pointer to w.

Hence, the path basically represents a binary search strategy. We want to investigate, how well this routing strategy can route arbitrary permutations. For this, consider some fixed permutation routing problem $\pi : U \to U$, and consider the random experiment of choosing a random hash function for the nodes.

Theorem 2.1 *π is routed by Chord with dilation at most $\log n$ and congestion at most $O(\log^2 n)$ w.h.p.*

Proof. First we show some claims. For a node $v \in V$ let

$$C_v = \{x \in [0,1) \mid v = \text{succ}(x)\} .$$

Claim 2.2 *For every node $v \in V$, $|C_v| \leq (2\log n)/n$ with high probability.*

Proof. Consider some fixed copy $v \in V$. The probability that there is no other node w with $h(w) \in [h(v) - (2\log n)/n, h(v)]$ is at most

$$\left(1 - \frac{2\log n}{n}\right)^{n-1} \leq e^{-(2\log n)(n-1)/n} < \frac{1}{n^2} .$$

Since there are only n nodes to consider, the claim follows. □

Claim 2.3 *For every interval $I \subset [0,1)$ of size $2/n$, there can be at most $6e\log n/\log\log n$ nodes with hash values in I w.h.p.*

Proof. Consider the set $\mathcal{I}$ of all intervals of the form $[k/n,(k+1)/n)$ for some $k \in \{0,\dots,n-1\}$. For any $I' \in \mathcal{I}$, the probability that there are at least k nodes with hash values in I' is at most

$$\binom{n}{k}\left(\frac{c}{n}\right)^k \leq \left(\frac{en}{k\cdot n}\right)^k = \left(\frac{e}{k}\right)^k .$$

Hence, if $k_0 = 2e\log n/\log\log n$, then this probability is at most n^{-2e}. Hence, the probability that there is any interval $I' \in \mathcal{I}$ with at least k_0 nodes is at most n^{1-2e}. Since any interval I of size $2/n$ overlaps with at most 3 intervals in $\mathcal{I}$, the claim follows. □

Next, we prove a bound on the dilation (which was already shown in a similar form in other papers).

Lemma 2.4 *The dilation of routing π is at most $\log n$ w.h.p.*

Proof. Consider any pair of nodes u, w with path $p(u,v) = (u = v_0, v_1, \dots, v_\ell = w)$. The binary search strategy and the definition of the fingers makes sure that for every j, $|h(v_{j+1}) - h(w)| \leq |h(v_j) - h(w)|/2$. Hence, for $j = \log n - 1$, $|h(v_j) - h(w)| \leq 1/2^{\log n - 1} = 2/n$. Using Claim 2.3, it follows that v_j must have a successor pointer to w w.h.p. if s is sufficiently large. Hence, the lemma follows. □

It remains to bound the congestion.

Lemma 2.5 *The congestion of routing π is $O(\log^2 n)$ w.h.p.*

Proof. First we introduce some notation. Consider any pair of nodes u and w. The *trajectory* $\tau(u,w)$ from u to w is a (potentially infinite) sequence of real numbers $x_0, x_1, x_2, x_3, \dots$ with the property that $x_0 = h(u)$ and for all $j \geq 0$, $x_{j+1} = x_j + 1/2^{i_j}$ where i_j is the smallest integer so that $x_{j+1} \leq h(w)$.

The next claim bounds the deviation of the path $p(u,w)$ from the trajectory $\tau(u,w)$. We assume here that for all $j \geq \ell$, $v_j = w$.

Claim 2.6 *For all $j \geq 0$, $|h(v_j) - x_j| = O((\log n)/n)$ w.h.p.*

Proof. Consider the set $\mathcal{I}$ of all intervals of the form $[x_0 + k/n, x_0 + (k+1)/n)$ (modulo 1) with $k \in \{0, \ldots, n-1\}$.

Suppose that there is a $j \geq 1$ with $|h(v_j) - x_j| \geq (c \log n)/n$. Then there must be disjoint intervals $J_1, J_2, \ldots, J_j$ consisting of consecutive intervals in $\mathcal{I}$ with J_g starting at x_g for every $1 \leq g \leq j$ so that $\sum_g |J_g| = ((c-1)\log n)/n$ and there is no node in $V \setminus \{u, w\}$ with $h(v) \in \cup_g J_g$. Since according to Lemma 2.4 we know that $j \leq \log n$, the probability that this happens is at most

$$\begin{aligned} \binom{(c-1)\log n + \log n - 1}{\log n}\left(1 - \frac{(c-1)\log n}{n}\right)^{n-2} &\leq \left(\frac{2ec\log n}{\log n}\right)^{\log n} \cdot e^{-(c-1)\log n \cdot (n-2)/n} \\ &\leq n^{\log(2ec)-(c-1)} \end{aligned}$$

which is at most $n^{-\gamma}$ for any constant γ if c is sufficiently large. Hence, the claim follows. □

Now, consider routing a matching $\mu \subseteq V \times V$, that is, every node is in exactly one pair $(u, w) \in \mu$. In this case, the endpoints of the trajectories $\tau(u, w)$ of the pairs $(u, w) \in \mu$ are independent because each node v has a value $h(v)$ chosen independently at random. We define a trajectory to be at stage j if it is at a point x_k with $|x_k - x_{k+1}| < 1/2^j$.

Claim 2.7 *For any interval I and any stage j, the expected number of trajectories of μ at stage j that are in I is equal to $|I| \cdot n/2$.*

Proof. We prove the claim by induction. Consider any interval $I \subseteq [0, 1)$. Certainly, the probability that a node v is in I is equal to $|I|$. Hence, the expected number of trajectories of pairs in μ that start in I (resp. that are at stage 0) is equal to $|I| \cdot n/2$.

Suppose now that we already showed that the expected number of trajectories at stage j that are in I is equal to $|I| \cdot n/2$ for any interval I. Then consider any interval J at stage $j+1$. Let $J_1 = (J - 1/2^{j+1}) \setminus J$, $J_2 = (J - 1/2^{j+1}) \cap J$, and $J_3 = J \setminus (J - 1/2^{j+1})$. Only trajectories that were in J_1, J_2, or J_3 at stage j can end up in J at stage $j+1$. Trajectories in J_1 end up in J is they jump ahead by $1/2^{j+1}$, trajectories in J_3 end up in J if they remain where they are at stage j, and trajectories in J_2 will always end up in J at stage $j+1$. Since the probability for a trajectory at stage j to jump ahead by $1/2^{j+1}$ is exactly $1/2$, the probability of a trajectory in J_1 or J_3 ending up in J is exactly $1/2$. As $|J_1| = |J_3|$ and $|J| = |J_2| + |J_3|$, the expected number of trajectories in J is equal to $|J| \cdot n/2$, which completes the induction. □

Since the trajectories are independent of each other, it follows from standard Chernoff bounds that for intervals I of size $(d \log n)/n$ for a sufficiently large d, at most $d \log n$ trajectories at stage j are in I w.h.p. Since due to Lemma 2.4 we only have to consider $\log n$ stages, the total number of trajectories with points in I is at most $d \log^2 n$ w.h.p.

From Claim 2.2 we know that for every node v, $C_v = (2 \log n)/n$ w.h.p., and from Claim 2.6 we know that paths can only deviate from their trajectories by an additive $(c \log n)/n$ w.h.p. Hence, only messages with trajectories τ that have a point x_j with $h(v) - x_j \leq ((c+2)\log n)/n$ will pass through v. Since there can only be at most $(c+2)\log^2 n$ such trajectories for any matching μ, any matching can be routed with congestion $O(\log^2 n)$ w.h.p. Since any permutation can be decomposed into two matchings, it follows that the congestion for routing π is $O(\log^2 n)$ w.h.p. □

Combining Lemma 2.4 and 2.5 yields the theorem. □

Let the nodes be numbered from 0 to $n-1$ and consider the $n-1$ permutations $\pi_i : x \to x + i \pmod n$, $i \in \{1, \ldots, n-1\}$. It follows from Theorem 3.16 that every π_i can be routed with dilation at most $\log n$ and congestion $O(\log n)$ w.h.p. Hence, when considering the system of paths $\mathcal{P}$ containing the routing path in Chord for every pair of nodes, $\mathcal{P}$ has a dilation of at most $\log n$ and a congestion of at most $O(n \log^2 n)$. This implies the following result concerning searchability:

Theorem 2.8 *Chord has a searchability of* $O(\log^2 n)$.

As an immediate corollary, we also get:

Corollary 2.9 *Chord has an expansion of* $\Omega(1/\log^2 n)$.

The results of the next section will imply an even better expansion.

3 Chord++

Next we show that when changing the overlay network and the routing strategy in a suitable way, we can even achieve a congestion of $O(\log n)$ w.h.p. We call the resulting system Chord++.

3.1 The overlay network of Chord++

The overlay network of Chord++ is very similar to Chord with the only difference every user u participating in the system has $c \log n$ *copies* u_j for some sufficiently large constant c, and each copy has $s \log n$ successor and predecessor pointers, called *neighborhood pointers*. Thus, the overall number of copies in the Chord++ system is $N = cn \log n$. This has the following consequence for the degree and the update cost for each user.

Claim 3.1 *Every user has a degree of* $O(\log^2 n)$ *with high probability, and inserting or removing a user from the Chord++ system takes* $O(\log^2 n)$ *work with high probability.*

The proof can be easily shown with the help of the techniques used in the proof of Theorem 3.2. Notice that also the original Chord system can only guarantee $O(\log^2 n)$ bounds with high probability since there exist nodes in Chord with a degree of $\Theta(\log^2 n)$ with high probability. This can happen if a node u does not have any other node in the area $[h(u) - (\log n)/n, h(u)]$, because then it will attract an expected number of $\log n$ nodes for each $1/2^i$ distance, resulting in $\Theta(\log^2 n)$ incoming edges with high probability.

3.2 Routing in Chord++

In this section we present a distributed online algorithm that can route any set of packets in Chord++ that form a permutation or a random routing problem with congestion and dilation $O(\log n)$ with high probability. This also has interesting consequences for the original Chord system. To avoid any ambiguities, we define the terms used in this statement.

A *permutation routing problem* is a routing problem in which every user is the origin and the destination of exactly one lookup operation. In a *random routing problem*, every user is the origin of exactly one lookup operation, and every lookup operation chooses a user uniformly and independently at random as its destination. Random routing problems model the situation where data is distributed uniformly at random among the users, as this has been suggested for Chord. So once we know how to route random routing problems, we can not only efficiently route lookup operations but also concurrent insert, delete, and join operations in Chord. Furthermore, being able to route arbitrary permutation problems makes sure that even if there is a correlation between the requests, it holds that as long as their sources and destinations distribute nicely, we can still perform routing in an efficient way. Hence, both permutation routing and random routing are important primitives that have to be supported efficiently.

Given a collection of routes R through the users, we define the *dilation* of R as the length of the longest route (i.e. the maximum number of hops in a route) and the *congestion* of R as the user with the largest number of routes crossing it. For lookup requests to be supported in an efficient way, it is important to ensure that both the congestion and dilation are small. It is well-known that the best upper bound that can be reached for the congestion and dilation of a random routing problem in a network of polylogarithmic degree is $O(\log n / \log \log n)$. We will present on online protocol for CHORD++ with an upper bound of $O(\log n)$ with high probability.

3.3 The online protocol

Let CHORD++ be the set of copies of users in the system. For every value $x \in [0, 1)$ we define

$$\text{succ}(x) = \mathsf{argmin}\{v \in \text{CHORD++} \mid h(v) \geq x\}$$

and for every range $R = [x, y] \subseteq [0, 1)$ and $\epsilon \in [0, 1)$ we define $R + \epsilon = [x + \epsilon, y + \epsilon]$.

Suppose that the users want to route an arbitrary permutation. I.e. given a permutation $\pi \in S_n$, user u wants to send a request to user $\pi(u)$. Then we use the following strategy for a request packet P from u to $\pi(u)$:

First, a random copy u_j is chosen for u and a random copy $\pi(u)_{j'}$ is chosen for $\pi(u)$. Let $x = h(u_j)$ and $z = h(\pi(u)_{j'})$. Choose a random real number $x' \in [x - (k \log N)/N, x + (k \log N)/N]$ and a random real number $z' \in [z - (k \log N)/N, z + (k \log N)/N]$. Also, a random real number $y \in [0, 1)$ is selected. In order to send p from u to $\pi(u)$, we go through the following stages:

1. p is sent from x to $\text{succ}(x')$ by using one of x's .
2. p is sent from $\text{succ}(x')$ to $\text{succ}(y)$ by using $\mathsf{Send}(x', y)$.
3. p is sent from $\text{succ}(y)$ to $\text{succ}(z')$ by using $\mathsf{Send}(y, z')$.
4. p is sent from $\text{succ}(z')$ to z by using one of $\text{succ}(z')$'s neighborhood pointers.

The function $\mathsf{Send}(x, y)$ works as follows:

Suppose that the current location of P is $\text{succ}(q)$ for some $q \in [0, 1]$. If there is a neighborhood pointer from $\text{succ}(q)$ to $\text{succ}(y)$ then take it. Otherwise, let i be the smallest integer so that $q + 1/2^i \leq y$. Then move from $\text{succ}(q)$ to $p_i(\text{succ}(q))$ and from there to $\text{succ}(q')$ with $q' = q + 1/2^i$ using a neighborhood pointer.

3.4 Analysis

The routing protocol has the following performance.

Theorem 3.2 *Every permutation π on the users can be routed by Chord++ with user-congestion and dilation $O(\log n)$ with high probability.*

Proof. First some definitions. For a copy $v \in$ CHORD++ we define

$$C_v = \{x \in [0, 1) \mid v = \text{succ}(x)\} ,$$

and for every range $R \subseteq [0, 1)$ let

$$C_R = \bigcup_{v \in \text{ Chord: } h(v) \in R} C_v .$$

Next some claims are given that will be helpful for the proof.

Claim 3.3 *For every copy $v \in$ CHORD++, $|C_v| \leq (2 \log N)/N$ with high probability.*

Proof. Consider some fixed copy $v \in$ CHORD++. The probability that there is no other copy w with $h(w) \in [h(v) - (2\log N)/N, h(v)]$ is at most

$$\left(1 - \frac{2\log N}{N}\right)^{N-1} \leq e^{-(2\log N)(N-1)/N} < \frac{1}{N^2} .$$

Since there are only N copies to consider, the claim follows. $\square$

Claim 3.4 *For any fixed range $R \subseteq [0,1)$ with $|R| = (r\log N)/N$ where $r \geq 1$ it holds that*

$$|\{u \in \text{CHORD++} \mid h(u) \in R\}| \leq 4r\log N$$

with high probability.

Proof. It holds that

$$\begin{aligned} \Pr[|\{u \in \text{CHORD++} \mid h(u) \in R\}| \geq 4|R| \cdot N] &\leq \binom{N}{4|R| \cdot N} \cdot |R|^{4|R|\cdot N} \\ &\leq \left(\frac{eN}{4|R| \cdot N}\right)^{4|R|\cdot N} \cdot |R|^{4|R|\cdot N} \\ &= \left(\frac{e}{4}\right)^{4|R|\cdot N} \leq \left(\frac{1}{2}\right)^{2r\log N} = \frac{1}{N^{2r}} . \end{aligned}$$

$\square$

The claims allow us to prove our first lemma about the protocol.

Lemma 3.5 *Each time the protocol uses a neighborhood pointer, a neighborhood pointer can indeed be used with high probability.*

Proof. If $s > 4k$, then it follows from Claim 3.4 that this is true for stages 1 and 4. So consider now simulating the transmission of a packet from $\text{succ}(q)$ to $\text{succ}(q')$ in Send. For this it is first sent from $\text{succ}(q)$ to $p_i(\text{succ}(q))$ for some i, where $p_i(\text{succ}(q)) = \text{succ}(h(\text{succ}(q)) + 1/2^i)$, and from there to $\text{succ}(q + 1/2^i)$. Claim 3.3 implies that

$$\begin{aligned} |\text{succ}(h(\text{succ}(q)) + 1/2^i) - \text{succ}(q + 1/2^i)| &\leq \\ |(q + (2\log N)/N + 1/2^i) + (2\log N)/N - (q + 1/2^i)| &= \frac{4\log N}{N} \end{aligned}$$

with high probability. Thus, if $s > 4 \cdot 4$, then the step from $p_i(\text{succ}(q))$ to $\text{succ}(q')$ can also be performed via a neighborhood pointer with high probability. $\square$

Hence, the protocol works correctly with high probability. Next we prove a bound on the dilation.

Lemma 3.6 *The dilation of the paths used by the protocol is $O(\log N)$ with high probability.*

Proof. From Lemma 3.5 it follows that stage 1 and 4 need one hop. Also, in each of stages 2 and 3, every i is used at most once because always the smallest i with $q + 1/2^i \leq y$ resp. z' is chosen (see the protocol). Furthermore, for each i a packet traverses at most two edges, namely from $\text{succ}(q)$ to $p_i(\text{succ}(q))$ and from $p_i(\text{succ}(q))$ to $\text{succ}(q + 1/2^i)$. Once an i is reached with $1/2^i \leq (s\log N)/(4N)$, it follows from Claim 3.4 that there must be a neighborhood pointer from $\text{succ}(q)$ to $\text{succ}(y)$ resp. $\text{succ}(z')$. Since every i is used at most once in a call of Send, it follows that the path length is $O(\log N)$ with high probability. $\square$

It remains to bound the congestion.

Lemma 3.7 *The user-congestion of the paths used by the protocol is $O(\log N)$ with high probability.*

Proof. Since each packet chooses a random copy of a user as starting point and endpoint, the expected number of packets originating and ending at a copy is $1/(c \log N)$. Consider any range $R = [a, b] \subseteq [0, 1)$ of size $(r \log N)/N$. Only those packets will have their x' in R that originate from copies v with $h(v) \in [a - (k \log N)/N, b + (k \log N)/N]$, and from Claim 3.4 it follows that the number of copies in this range is at most $4(2k + r) \log N$ with high probability. Hence, the expected number of packets that choose a starting point $x' \in R$ is at most

$$(4(2k + r) \log N) \cdot \frac{1}{c \log N} \cdot \frac{|R|}{(2k \log N)/N} = \frac{4(1 + r/(2k))|R| \cdot N}{c \log N} \leq \frac{|R| \cdot N}{\log N} \quad (1)$$

if $c \geq 4(1 + r/(2k))$. The same holds for the expected number of packets that choose their endpoint $z' \in R$.

Since y is chosen uniformly at random for each packet, it holds:

Claim 3.8 *For every current location q of a packet during* $\mathsf{Send}(x', y)$ *and every i for which $1/2^{i'}$-jumps have already been considered with $i' < i$, the probability of going from q to $q + 1/2^i$ or staying at q (because $|q - y| < 1/2^i$) is exactly $1/2$.*

Proof. If all i' with $i' < i$ have already been considered, then it must hold that the packet is at location q with $|q - y| < 1/2^{i-1}$. Since y is chosen uniformly at random, the probability that $|q - y| < 1/2^i$ is equal to the probability that $|q - y| \in [1/2^i, 1/2^{i-1})$. □

This implies the following claim.

Claim 3.9 *For every range $R \subseteq [0, 1)$ of size δ it holds: For every i the expected number of packets in R after considering all $i' < i$ in* Send *is at most $\delta N/\log N$.*

Proof. The claim is shown by induction. It follows from (1) that before $i = 1$ the expected number of packets in R is at most $\delta N/\log N$. Thus, assume that this is true before some arbitrary i. Then it follows from Claim 3.8 that the expected number of packets in R after considering i is at most

$$\frac{1}{2} \cdot \frac{\delta N}{\log N} + \frac{1}{2} \cdot \frac{\delta N}{\log N} = \frac{\delta N}{\log N} ,$$

where the first term comes from those packets that stay in R and the second term comes from packets in range $R' = R + 1/2$ that decide to move to position $q + 1/2^i$. This concludes the proof. □

Notice that the same result also holds for $\mathsf{Send}(y, z')$.

Hence, in a continuous setting, we get an evenly balanced congestion throughout the whole routing. It remains to bound the congestion for simulating this in the discrete setting of Chord. From Claim 3.9 it follows that the expected congestion at copy v for some fixed i caused by having $v = p_i(\mathrm{succ}(q))$ for some q is at most

$$\frac{|C_{C_v - 1/2^i}| \cdot N}{\log N}$$

and the expected congestion at copy v caused by having $v = \mathrm{succ}(q')$ for some $q' = q + 1/2^i$ is at most

$$\frac{|C_v| \cdot N}{\log N} .$$

Hence, the total expected congestion (w.r.t. the random packet choices) at a copy $v \in$ CHORD++ during an execution of Send is at most

$$\left(\sum_{i=1}^{\log(N/((s/4)\log N))} \frac{|C_{C_v - 1/2^i}| \cdot N}{\log N}\right) + \log N \cdot \frac{|C_v| \cdot N}{\log N} \ . \tag{2}$$

In order to bound $\sum_i |C_{C_v-1/2^i}|$ we need the following claim.

Claim 3.10 *For every copy v it holds that*

$$\sum_{i=1}^{\log(N/((s/4)\log N))} |C_{C_v-1/2^i}| \leq |C_v| \log N + (5 \log N)/N$$

with high probability.

Proof. Consider some fixed copy v, and let u be its predecessor on the base cycle (i.e. $C_v = [h(u), h(v)]$). Let $x_i = h(u) - 1/2^i$ for all $i \in \{1, \ldots, \log(N/((s/4)\log N))\}$ and let $R_i = [x_i - \delta_i, x_i]$ for some δ_i. The probability that there is a set of integers $k_i \geq 0$ with $\sum_i k_i = \alpha \log N$ so that for $\delta_i = k_i/N$ there is no copy in the ranges R_i is very low, because

$$\binom{\alpha \log N + \log N - 1}{\log N}\left(1 - \frac{\alpha \log N}{N}\right)^{N-2} \leq ((\alpha+1)e)^{\log N} \cdot e^{-(\alpha \log N)(N-2)/N} \leq \frac{1}{N^{\alpha/2}}$$

if α is sufficiently large. Hence, $\sum_i |[h(\text{pred}(x_i)), x_i]|$ (where $\text{pred}(x_i)$ is the copy closest to x_i from below) can be at most $((\alpha+1)\log N)/N$ with high probability, because otherwise sets R_i with $\delta_i = k_i/N$ and $\sum_i k_i = \alpha \log N$ can be identified with no copy in them. Since $C_{C_v - 1/2^i} = [h(\text{pred}(x_i)), x_i] \cup [h(u) - 1/2^i, h(v) - 1/2^i]$ it follows that

$$\sum_{i=1}^{\log(N/((s/4)\log N))} |C_{C_v-1/2^i}| \leq \frac{(\alpha+1)\log N}{N} + \log N \cdot |C_v|$$

with high probability. Because there are only N copies v to consider, the claim follows. □

Claim 3.10 implies that (2) is at most

$$\frac{(|C_v| \log N + (5 \log N)/N) \cdot N}{\log N} + |C_v| \cdot N = 2|C_v| \cdot N + 5$$

with high probability. Summing up over all copies representing a single user u, it follows that u has an expected congestion of at most

$$\left(2N \sum_j |C_{u_j}|\right) + 5c \log N \ .$$

The next claim bounds $\sum_j |C_{u_j}|$.

Claim 3.11 *For every user u, $\sum_j |C_{u_j}| \leq (4c+1)(\log N)/N$ with high probability.*

Proof. Consider some fixed user u. The probability that there is a set of integers $k_j \geq 0$ with $\sum_j k_j = 4c \log N$ so that for ranges C_{u_j} of size $\delta_j = k_j/N$ there is no copy in C_{u_j} for all j is very low, because

$$\begin{aligned}\binom{4c\log N + c \log n - 1}{c \log n}\left(1 - \frac{4c \log N}{N}\right)^{N - 4c\log n} &\leq \left(\frac{5e \cdot c \log N}{c \log n}\right)^{c \log n} \cdot e^{-\frac{4c \log N}{N} \cdot (N - 4c \log n)} \\ &\leq e^{3c \log N} \cdot \mathrm{e}^{-3.9c \log N} \leq N^{-c} \ .\end{aligned}$$

Hence, $\sum_j |C_{u_j}| \leq (4c+1)(\log N)/N$ with high probability because otherwise sets C_{u_j} with $\delta_j = k_j/N$ and $\sum_j k_j = 4c \log N$ would exist with no copy in them. $\square$

Hence, the expected congestion at user u is at most $10c \log N$. On the other hand, one can show the following claim.

Claim 3.12 *For any packet P and user u it holds: P visits $\cup_j C_{u_j}$ at most 3 times along non-neighborhood pointer edges with high probability during stage 2 or stage 3 of the routing algorithm.*

Proof. W.l.o.g. let us concentrate on stage 2. First, suppose that P moves to a copy in the same set C_{u_j} for two different distances $1/2^{i_1}$ and $1/2^{i_2}$. Then it must hold that $i_1 = i_2 - 1$ and, according to Claim 3.3, $1/2^{i_2} \leq (2 \log N)/N$. However, in this case y would have only been a distance of at most $(8 \log N)/N$ away before considering distance $1/2^{i_1}$. Thus, if k is large enough, a neighborhood pointer could have been used immediately to send P to y, contradicting our assumption. Hence, P only traverses once a finger into any of the sets C_{u_j}. Since according to Claim 3.3 $|C_{u_j}| \leq (2 \log N)/N$ with high probability and the $h(u_j)$ are chosen independently at random, the probability that P moves at least 4 times into a set C_{u_j} in stage 2 or 3 is at most

$$\binom{\log N}{4}\binom{c \log N}{4} \cdot \left(\frac{2 \log N}{N}\right)^4 \leq \frac{c^4 (\log N)^{12}}{N^4} \ ,$$

which is sufficiently small so that for no packet and user combination this situation will be encountered with high probability. $\square$

Hence, with high probability, every packet will only contribute a constant value of at most some γ to the congestion at u. Since the random choices of the packets are independent of others, we can use the following theorem (see [2]):

Lemma 3.13 *Let $X_1, \ldots, X_n$ be independent random variables with $X_i \leq b$ for every i. Furthermore, let $X = \sum_{i=1}^n X_i$ and μ be chosen so that $\mu \geq \mathrm{E}[X]$. Then it holds for every $\delta > 0$ that*

$$\Pr[X \geq (1+\epsilon)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu/b} \ .$$

Hence, defining X_i as the contribution of packet i to the congestion of user u, it follows from the fact that $\mathrm{E}[X] \leq 10c \log N$ that $X = O(\log N)$ with high probability, i.e. the congestion at u is $O(\log N)$ with high probability. $\square$

Combining the lemmas yields the proof of the theorem. $\square$

A direct extension of the proof also implies the following result.

Corollary 3.14 *Every routing problem in which the user is the source and destination of at most p packets can be routed with congestion $O(p \log n)$ and dilation $O(\log n)$ with high probability.*

The result for $p = c \log n$ (i.e. each copy has on average one packet) also implies the following corollary.

Corollary 3.15 *Every permutation π on the copies can be routed with user-congestion and dilation $O(\log^2 n)$ with high probability.*

Hence, when applying our routing protocol to the original Chord network, we get:

Theorem 3.16 *Every permutation can be routed in Chord with congestion $O(\log^2 n)$ and dilation $O(\log n)$ with high probability.*

Thus, one can also route efficiently in the original Chord network, which has the advantage that work for joining and leaving the system is on average much cheaper than for Chord++, but at the expense of increasing the congestion for routing. The result of Theorem 3.16 is actually tight since it can be shown that our protocol will cause some nodes in Chord to receive $\Theta(\log^2 n)$ packets with high probability, namely those that occupy a range of size $\Theta((\log n)/n)$ in the $[0, 1)$-ring.

Finally, we note that the proof of Theorem 3.2 can also be applied to random routing problems, because the expected number of requests heading for a particular user is 1, and therefore the bound on the expected congestion in the proof is still valid. In fact, since the destination is already chosen at random, one does not need a random intermediate location y any more to obtain the following result:

Corollary 3.17 *A random routing problem on the users can be routed in Chord++ with congestion and dilation $O(\log n)$ with high probability.*

Since permutation routing and random routing problems can be supported efficiently, concurrent lookup, insert, and delete operations can also be routed efficiently.

3.5 Concurrent join operations

Next we show that the proof of Theorem 3.2 can also be used to bound the congestion for concurrent join operations. Note that concurrent join operations are more difficult than lookup operations because after the new users have been forwarded to their correct locations, they have to be integrated into the system by establishing and updating fingers and neighborhood pointers, which also creates congestion. The integration is done concurrently for all new copies u of users by going through the following stages:

1. Every new copy u is integrated into the Chord cycle. For example, if old nodes v and w (i.e. v and w are already in Chord) are the predecessor resp. successor of the new node u, then the edge (v, w) is replaced by the edges (v, u), (u, w).
2. Every new copy u sends a message to its $s \log n$ predecessors and successors asking them to contact u for the establishment of neighborhood pointers. This also allows old copies already in the system to update their fingers: a copy w receiving a request from u that finds out that a finger from v to it should instead go to u, will notify v about that. Furthermore, new copies to get to know the fingers of the closest successor that was in the system before: if the old node v is the first successor of u, then v sends u its finger table.
3. Every new copy u uses the fingers from the old copy being its closest successor to establish its own fingers.

Theorem 3.18 *New users with random locations for their copies, one new user per old user, can be integrated into Chord++ with user-congestion $O(\log^2 n)$ and dilation $O(\log n)$.*

Proof. (Sketch) If each user has one new user, then each user has to send out $c \log n$ requests to integrate copies. According to Corollary 3.14, this can be done with congestion $O(\log^2 n)$ and dilation $O(\log n)$ with high probability. Hence, it remains to consider the stages above to integrate copies into the system.

The integration of new copies into the Chord cycle and the establishment of neighborhood pointers obviously only need congestion $O(\log^2 n)$. Updates of existing fingers in stage 2 only create a congestion of $O(\log^2 n)$ because the maximum degree of a user in the system is $O(\log^2 n)$. Finally, also stage 3 needs user-congestion at most $O(\log^2 n)$ because one can show with the proof methods of Theorem 3.2 that every old Chord++ user is contacted by at most $O(\log^2 n)$ new users with high probability. □

Note that the congestion bound is tight because users in Chord++ have a degree of $\Theta(\log^2 n)$ with high probability. The theorem immediately implies the following result.

Corollary 3.19 *New users with random locations, one per old user, can be integrated into Chord with congestion $O(\log^2 n)$ and dilation $O(\log n)$.*

4 Fault tolerance

Given a graph $G = (V, E)$, the *(node) expansion* $\alpha(G)$ of G is defined as

$$\alpha(G) = \min_{U \subset V,\, |U| \le |V|/2} \frac{|\Gamma(U)|}{|U|}$$

where $\Gamma(U) \subseteq V \setminus U$ is the set of all nodes outside of U to which a node in U has an edge. It is straightforward to see that the larger the expansion, the more difficult it is to disconnect a set U from the rest of the system, because at least $\alpha(G)|U|$ node failures would be necessary for this. Hence, the expansion is an important measure for the quality of a peer-to-peer system. There is a close relationship between the expansion and the congestion of random routing problems, expressed in the following result.

Lemma 4.1 *For every graph G in which a random routing problem can be handled with congestion C with high probability, $\alpha(G) = \Omega(1/C)$.*

Proof. Suppose that there is a set $U \subseteq V$ with $|U| \le |V|/2$ and $|\Gamma(U)| = o(|U|/C)$. Then consider a random routing problem in G. It is easy to see that the expected number of packets that have to move into and out of U is equal to $|U|$, and this is at least $|U|/2$ with high probability. On the other hand, we know that a random routing problem can be routed in G with congestion C with high probability. However, this is not possible to achieve with $|U|/2$ packets if $|\Gamma(U)| = o(|U|/C)$. □

Hence, Corollary 3.17 implies the following result.

Theorem 4.2 *Chord++ has an expansion of $\Omega(1/\log n)$.*

Hence, also the following is true.

Corollary 4.3 *Chord has an expansion of $\Omega(1/\log n)$.*

5 Conclusion

In this paper we demonstrated that if users are given sufficiently many nodes in Chord, then permutation routing and random routing problems can be handled online with congestion and dilation $O(\log n)$, with high probability. This result has many interesting consequences. It implies that also in the original Chord system, concurrent requests can be served efficiently, though the congestion deteriorates to $O(\log^2 n)$. Furthermore, it implies that Chord has a node expansion of $\Omega(1/\log n)$. The congestion matches the bound shown for Viceroy, and the expansion matches the bound shown for skip graphs, both of which have been the best published bounds obtained for peer-to-peer systems so far. Hence, Chord can compete well with other proposed peer-to-peer systems.

References

[1] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

[2] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[3] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of the 29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, 1997.

[4] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[5] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA Proceedings*, 2003.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, San Diego, CA, 8 2001.

[7] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 11 2001.

[8] 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*, San Diego, CA, 8 2001.

[9] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Theory of Computing*, pages 263–277. ACM, May 1981.

[10] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *UCB Technical Report UCB/CSD-01-1141*, 2001.