Group Protocols for Peers-Based Systems: A Case Study

Hui Fang[†], Wen Jing Hsu[‡], and Larry Rudolph[§]

[†] Singapore-MIT Alliance, National University of Singapore

[‡]Nanyang Technological University, Singapore

[§]CSAIL, MIT

email: {fanghui@nus.edu.sg;hsu@theory.csail.mit.edu;rudolph@lcs.mit.edu}

October 2005

Abstract

Group protocols are needed under circumstances where a large number of peers may join or leave the system simultaneously. For instance, collaborative groups of peers may form, merge or split dynamically. Without a group-oriented protocol coordinating the transformation of topology, the simultaneous joining/leaving of a massive number of nodes can impair the routing information and hence the system functionality. Existing P2P protocols only allow nodes to join or leave individually which unfortunately either causes a large number of messages in the process or results in long delays before the system can function normally.

Here we present two protocols for coordinated merging of Chord overlays. The goal is to merge all the nodes in two given Chord overlays into a single new Chord ring, while re-configuring their routing tables to support correct and efficient searching. In order to reduce the number of messages and time delays, both protocols take advantage of the existing structures of the overlays. Our methods also allow concurrency in the merging operations, which can reduce the overall latency. Let M and N denote the number of nodes in the two given overlays, where $M \leq N$. Our improved protocol requires no more than $O(M \lg \frac{M+N}{M} \lg(M+N))$ messages and an overall latency of $O(\lg \frac{M+N}{M} \lg(M+N))$ time steps.

1 INTRODUCTION

In the past few years, many peer-to-peer (P2P) systems based on DHTs (Distributed Hashing Tables) have been proposed, e.g. [1, 2, 3, 4]. These systems are both decentralized

and autonomous, and the protocols can work with individual node arrivals and departures. However, few protocols are currently available for coordinating group actions such as the merging of application-level network overlays. One exception may be [12] where protocols for forming subgroups from a given Chord are presented. Two P2P systems may need to be merged into a single new P2P system under various circumstances. For example, in computing grid based on dynamic P2P system, two well-structured subgroups may need to form a single group to provide more powerful computing capability. Thus merging operation may also be used to combine ad hoc collaborative groups on the computational Grid [13].

Here we will consider the specific problem of merging two Chord overlays into one. The goal is to reassign all the nodes onto the new Chord along with their routing tables (called the *finger tables*) to support correct and efficient searching. In Chord, at least $\Omega(N \lg^2 N)$ messages and the same order of time delays will be required, where N denotes the total number of nodes in the combined network. Stoica et al. [1] prove that when less than N nodes join a stable Chord with N nodes, if all successor pointers (but perhaps not all the other fingers) are correct, then lookups will still take $O(\lg N)$ time with high probability. However, every time after N nodes joining, $O(\lg^2 N)$ rounds of stabilization have to be done to correct the states. The protocol entails up to $O(\lg^2 N)$ messages overall. So far there is no protocol guaranteeing the stability of the system when a large number of nodes all join or depart simultaneously (as is the case with merging). Libe-Nowell et al. [11] analyzed the maintenance rate of N-node Chord with appendage states, and the proposed join model also showed that when N nodes join the network concurrently, after at least $\lg^2 N$ rounds of stabilization the lookup service can be achieved with high probability. Karger and Ruhl [12] proposed a Diminished Chord for group formation in P2P networks, which allows subgroups of nodes to form in the network by adding extra finger entries to the subsets of nodes. The group protocol entails $O(\lg^2 N)$ rounds of stabilization and $O(N \lg^2 N)$ messages.

The existing join and departure protocols could be applied to handle the merging process by firstly dispersing all the peers using a *departure* protocol, who then join again to form a new overlay by using a *joining* protocol. However, the process generally entails a large number of messages and a long overall latency before the new system can start functioning. We will show that the merging process could be substantially eased by coordinating the join/leave process and by exploiting the existing structures in the overlays. For instance, instead of forming the target Chord from scratch, we may choose one existing overlay as a base and let the peers in the second chord join this base. In the merging process, the existing finger entries in the overlays could also be reused as much as possible to set up the new overlay. This coordinated approach could save communication bandwidth and ensure a correct merged result.

To illustrate our point, we will first consider a preliminary method for the merging of two Chord overlays by concatenating them together, retaining a major portion of the finger entries. Our analysis of the protocol shows that the message and the delays are indeed reduced by a $O(\lg N)$ factor, where N denotes the total number of nodes in the combined

networks. However, there are certain restrictions about this approach. In our second, improved protocol, we then show how to carry out even more efficiently. The process consists of two short phases. The first phase will combine two Chords into a relaxed, but sufficiently near version of Chord; the second phase then further refines it. The merging process takes substantially fewer messages and causes shorter delays. The second protocol also allows parallelism in the merging operations. Let M and N denote the number of nodes in the two merging Chord overlays, where $M \leq N$. Our improved protocol requires no more than $O(M \lg \frac{M+N}{M} \lg(M+N))$ messages and an overall latency of $O(\lg \frac{M+N}{M} \lg(M+N))$ time steps.

The contributions of this paper are thus two-fold: (a) the provably efficient algorithms for concurrent merging of two overlays, and (b) the proofs of the message complexity. We also present Relaxed Chord, a generalized version of Chord, which does not require the fingers to point at the exact location, but allows them to "float" within a certain range. This relaxation allows us to reduce the messages required for readjusting the fingers during the merge, yet it still can guarantee to support lookups within $O(\lg N)$ hops.

The rest of the paper is organized as follows. Section 2 describes the model and formulation. Section 3 presents the protocols for merging Chords and proves their correctness. In Section 4, we summarize our results and suggest issues for future research.

2 MODEL AND FORMULATION

Chord uses a Distributed Hash Table(DHT) to support the search function. The resource objects (as identified by unique keys) and the hosts that maintain these resources are assigned IDs in the same *identifier space* which forms a ring modulo 2^n. Object k [1] is assigned to the first host whose identifier equals to or follows k in the identifier space. This host is called the *successor* of key k, denoted by *successor(k)*. A node y is said to be *x's predecessor* node, *iff* (if and only if) x is y's successor node.

Each host in Chord maintains a finger table which records at most $O(\lg N)$ other peer addresses. For a node x, *finger[i]* in x is defined as the first node on the circle that succeeds $(x + 2^{i-1})$ *mod* 2^n, i.e. $finger[i] = successor(x + 2^i)$, where $1 \leq i \leq n$. Hence *finger[1]* points to *x's successor* node on the identifier circle. Chord provides only one basic function *lookup(key)* that yields the node responsible for the key.

We write $X = Chord(N, n, h)$ to denote a Chord X with ID space equal to 2^n, a node set $X_N := \{x_i : 0 \leq i < N\}$ and a hash function $h : X_N \to \mathbb{N}$ which maps the node set X_N to the identifier space of Chord X. Where there is no danger of confusion, the node set X_N may also be identified by X. When there is only one Chord $X = Chord(N, n, h)$ in the context, X may be written as $X = Chord(N, n)$, and a node x in X is written as $x \in X$.

[1] To simplify our notations, where there is no danger of confusion, we identify the ID with the hashed value. Thus, here k denotes the hashed value of the given object.

Given two Chords $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$, our goal is to merge them into one larger Chord $Z = Chord(R, r, id_Z)$. So the new Chord Z will have $R = M + N$ nodes in total. Furthermore, the newly generated Chord will maintain connection information defined by the finger tables of all peers. One immediate issue is to decide the ID space of Chord Z. Clearly, the dimension of Z must satisfy the following inequality: $r \geq max(n, m, lg(M + N))$. We assume $m \leq n$ without loss of generality. In other words, r could be chosen to equal n or $n + 1$. $r = n$ means that all peers in Chord Y will be merged into X's current ID space. In this case, Chord X still has enough vacancies in its ID space, i.e. $\lg(M + N) \leq n$. The case $r = n + 1$ means that the new Chord has two times identifier space larger than X. There is no major difference between the two cases whether r equals to n or $n + 1$, because section 3.3 shows that, by shifting the table entries, we can easily enlarge X's ID space by two times (or 2^k times for any $k \geq 1$).

In the remaining part of the paper, unless otherwise mentioned, we assume that, before merging, the identifier assignments of the nodes on each Chord are distributed uniformly at random on the respective ID space; moreover, the information stored in the finger tables of each Chord is correct; also, during the merging procedure, there is no node joining or leaving or node/link failures.

3 APPROACH AND CORRECTNESS PROOF

As mentioned earlier, it is straightforward to construct a new Chord by inserting the nodes one by one. However, to conserve on the number of messages and to reduce the overall latency, it makes more sense to preserve the structures of the existing chords as much as possible.

To illustrate a coordinated merging process, we show the following protocol (Protocol 1) which combines two Chord overlays to produce a new Chord.

3.1 A protocol for coordinated merging

The key idea here is for Chord X and Chord Y to each occupy a non-overlapping half space of Z. Based on the hash function id_X and id_Y, the identifier assignment on Z is given by:

$$\forall z \in Z, id_Z(z) := \begin{cases} id_X(z) \ if \ z \in X \\ id_Y(z) + 2^n \ if \ z \in Y \end{cases}$$

3.1.1 Merging procedure

The merging can start with any pair of nodes $x_{src} \in X$ and $y_{src} \in Y$, as long as there is a connection setup between x_{src} and y_{src}. x_{src} updates its finger table via y_{src}, and it

will prompt the other nodes in X to update (modify or create new) fingers. The protocol (*Protocol 1*) is shown in Table 1.

On node x_{src}: //Node y_{src} searches the successor of key $id_X(x_{src})$ in Y $y := y_{src}.lookup(id_X(x_{src}))$; finger[$n$+1]:= y; **for** i=1 to n **if** $id_X(x_{src}) + 2^{i-1} \geq 2^n$ **do**   finger[i]:= $y.lookup(id_X(x_{src}) + 2^{i-1} - id_Y(y))$;   **endif** **endfor** send a message with content $< x_{src}, y,$"$merge$" $>$ to its successor; wait until the merge message returns from its predecessor and broadcast to all to terminate.
On any other node x: receive the message with content $< x_{src}, y,$"$merge$" $>$ from its predecessor; **if** $(id_X(x) > id_Y(y))$ **do**   //y is replaced by the successor of key $id_X(x)$ in Y, which is found by node y $y := y.lookup(id_X(x))$; **endif** finger[n+1]:= y; **for** i=1 to n **if** $id_X(x) + 2^{i-1} \geq 2^n$ **do**   finger[i]:= $y.lookup(id_X(x) + 2^{i-1} - id_Y(y))$;   **endif** **endfor** send a "readjust" message with content $< x_{src}, y,$"$merge$" $>$ to its successor;

Table 1: Protocol 1

x_{src} first tells its successor to start the merging operations, which in turn informs its successor to proceed in the same way and so on. Nodes in Chord Y will carry out a corresponding sequence of actions to adjust their finger tables. When a node x receives the message, it will update its finger table. The update action is composed of adding a new entry finger[n+1] and modifying possibly out-of-date fingers. By unraveling the incoming message with information about its peer y, x can then set fingers to point at nodes in the second Chord. The process stops when the "merge" message returns to x_{src} and y_{src}.

3.1.2 Changes in finger entries

Here we analyze the number of finger table entries which need to be added or modified for both Chords. With regard to the newly added entries (arising from the very last entry of each finger table), we found:

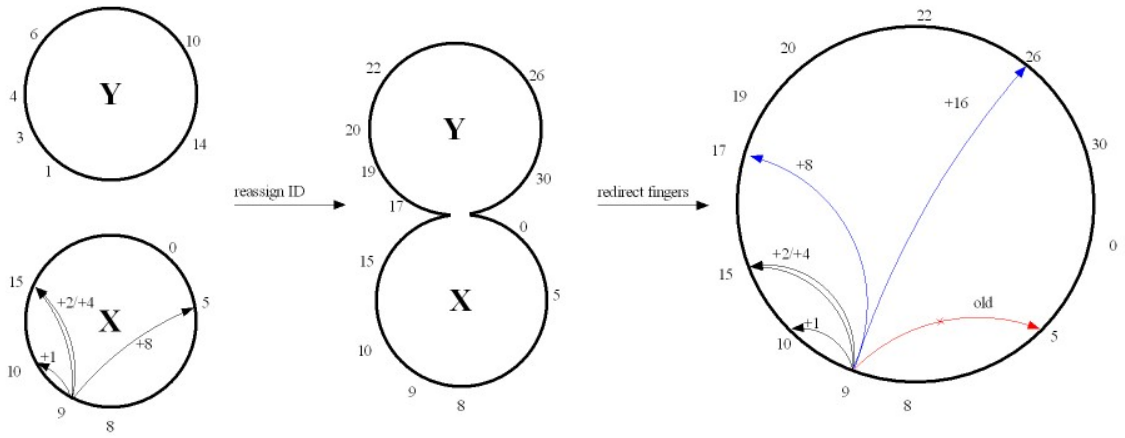

Figure 1: Merging two Chord rings that are of the same dimension

Lemma 1: Let $X = Chord(N, n, id_X)$ and $Y = Chord(M, n, id_Y)$ denote two Chord networks to be merged by using Protocol 1 (as given in Table 1) and the resulting overlay is Z. Then $Z = Chord(M + N, n + 1, id_Z)$. Also, (1) the total number of *different* new fingers and the number of times executed of the lookup operation are both no more than $2min(N, M)$. (2) For each lookup $y.lookup(id_X(x))$ in Table 1, $|id_X(x) - id_Y(y)| \leq |id_X(x) - id_X(x.predecessor)|$.

Proof. (Please refer to Appendix A.) □

It is well known that the the larest *zone* size, i.e. the maximun difference between two adjacent node IDs [5] [6], of a Chord network with N nodes is $\Theta(\lg N)$ times of the average zone size with high probability. Therefore the following corollary is immediate.

Corollary 1: with probability $1 - N^{-\epsilon}$, where $\epsilon > 0$ denotes a constant, each lookup $y.lookup(id_X(x))$ in Protocol 1 requires to contact no more than $n - \lg \frac{N}{\lg N} + \lg(1 + \epsilon)$ nodes in Chord Y.

Proof. (Please refer to Appendix B.) □

Besides adding one new entry to the finger table, we note that the existing fingers may become useless in the context of the new Chord. For instance, node 9 in Fig. 1 had a finger to node 5, which becomes invalid after merging. The new finger now points to node 26 (which was previously node 10 in Y). The following theorem offers an upper bound on the total number of modified entries.

Theorem 1: Let $X = Chord(N, n, id_X)$ and $Y = Chord(M, n, id_Y)$ denote two Chord networks to be merged by using Protocol 1 and let $Z = Chord(M + N, n + 1, id_Z)$ denote

the new Chord. Then the merging process requires to create less than $2 \cdot min(N, M)$ new fingers in Z, and to modify no more than $2(M + N) - \lg(N \cdot M) - 4$ fingers from X and Y.

Proof. (Please refer to Appendix C.) □

Since each lookup can be resolved via $O(lgN)$ messages to the other nodes on an N-node Chord, the total number of messages required for merging purpose is $O(N \lg N)$. Compared with a brute force method where each peer joins separately at a total cost of $O(N \lg N \lg N)$, the above method is more attractive. However, the approach mentioned above also suffers from a major drawback: the resulting Chord will have uneven node distribution on its ID space if there is a large difference between the number of nodes in Chord X and Y. Moreover, the overall completion time $O(N \lg N)$ is too long for large N. To solve this problem, we will introduce a new approach, which will be described next.

3.2 Modified Protocol

Without loss of generality, we assume that Chord Y has a lower dimension than X which also has a sufficiently large identifier space to accommodate all nodes. (Section 3.3 shows how to enlarge the identifier space without any communication cost.) In other words, we have $m < n$ and $\lg(M + N) \leq n$ as two initial conditions.

This protocol will merge Y into X. The identifier reassignment in Z is as follows:

$$\forall z \in Z, id_Z(z) := \begin{cases} id_X(z) \ if \ z \in X \\ id_Y(z) * 2^{n-m} + \delta \ if \ z \in Y \end{cases}$$

where δ denotes the least non-negative number which ensures that no other nodes have the same ID.

3.2.1 Merging procedure

Definition 1: Let $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$ be as defined before, where $m < n$. For each node y in Y, *Interval(y)* is defined by $[\ id_Y(y) * 2^{n-m}, (id_Y(y) + 1) * 2^{n-m}\ )$.

Protocol 2 is described in Tables 2, 3, and 4.

We will also use the following notations in the subsequent discussions:

$p = \frac{1}{2^n}$
$\beta = \frac{pN}{1-p}$
$\pi = 2^{n-m}$

1. broadcast merging message in Y.
2. each peer z in $Z = X \bigcup Y$ does: 2.1 z.reAssign(); 2.2 z.newFingerTabSetup();

Table 2: Protocol 2

```
// reassign an ID to node z
procedure z.reAssign( )
    if (z ∈ X)          return   id_X(z);
    id = id_Y(z) * 2^{n-m}; // if z ∈ Y
    while ( id == id_X(lookup(id)))        id++;
    return id;
end
```

Table 3: ID reassignment

```
// returns a new finger table with size n.
procedure z.newFingerTabSetup( )
    if (z ∈ X)
        newFingers= z.fingers; // copy the previous finger table.
    else // z ∈ Y
        newFingers= new(fingerTable, size=n);
        for (i = 1 to m) do
            copy( z.fingers+i, newFingers+n − m + i);
        endfor
         //Locally update in finger table
        for (i = 1 to n − m) do
           newFingers[i] = lookup(z + 2^i);
        endfor
    endif
    return newFingers;
end
```

Table 4: Create new finger table

It can be seen that π is the size of Interval(y) and β reflects the node saturability of Chord X.

One immediate question is: what is the probability that a peer from Y will collide with the identifiers of the existing peers in X? Lemma 2 shows that whether y can be inserted to Interval(y) or not depends on the parameters β and π.

Lemma 2: Suppose that Protocol 2 (as described in Tables 2-4) is applied to merge $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$ where $m < n$, and $\lg(M + N) < n$. For any

node y in Chord Y, (1) y's new ID in Z, $id_Z(y)$, is less than $id_Y(y) * 2^{n-m} + \frac{\beta}{(1+\beta)(1-\beta)^2}$ on expectation. (2) Let c denote a constant, where $0 \leq c < min(\pi, N)$. With probability $1-\beta^c$, y can be inserted to within the first c points of Interval(y).

Proof. (Please refer to Appendix D.) □

Lemma 2 shows that the success of merging a node y into a target interval depends on how full we have filled the peers in interval(y) and how large the interval is.

Theorem 2 follows directly from Lemma 2.

Theorem 2: Suppose that Protocol 2 (given in Tables 2-4) is applied to merge $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$ where $m < n$, and $\lg(M + N) < n$, and $n > (1 + \epsilon) \lg N$, $\epsilon > 0$. Then for any node $y \in Y$, with probability $1 - \frac{1}{N^{c\epsilon}}$, y can be inserted to within the first c points of the $Interval(y)$, where c denotes a constant and $0 < c < \min(\pi, N)$. □

The following proposition assesses the probability of Y being merged into X as a whole.

Theorem 3: Given $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$. If $n > m$ and $n \geq (2 + \epsilon) \lg N$, $\epsilon > 0$, then with probability $1 - N^{-\epsilon}$, Chord Y can be merged into Chord X, with each y in Y assigned an ID inside *Interval(y)*.

Proof. (Please refer to Appendix E.) □

Remarks: The precondition $n \geq (2 + \epsilon) \lg N$ in Theorem 3 seems a little pessimistic. Actually when the number of nodes in Y is also relatively small compared to X, it is with probability at least $1 - 2/N$ that each y in Y can be inserted to Chord X with an ID inside its $Interval(y)$. Applying Lemma 3.3 in [10], we obtain the following:

Corollary 4: Let $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$ be two given Chord overalys. If $n > m$ and $n \geq (\lg N + \lg(1 + \epsilon))$, and $m < \lg N + 2 \lg \epsilon - 3 - \lg \lg N$, where $\epsilon > 0$, then with probability at least $1 - 2/N$, Chord Y can be merged into Chord X with each y in Y assigned an ID inside *Interval(y)*.

3.2.2 Changes in fingers

Here we consider the efficiency of the modified protocol, in terms of the time required for the adjustment to the finger tables.

We will relax on the definition of Chord.

Definition 2: $X = Chord(N, n)$ is an α-Relaxed Chord iff: $\exists 0 < \alpha < 1, \forall x \in X$, $x.finger[i] = successor(x + 2^{i-1} + \delta_{x,i})$, where $\delta_{x,i} \in [0, \alpha 2^{i-2})$, $\quad 1 \leq i \leq n$.

Obviously, according to the definition, finger[1], finger[2] and finger[3] always reflect the exact pointers to the other nodes. Other than these, the remaining fingers can have multiple choices. To distinguish, we call the Chord defined in [1] the exact Chord, and the Chord by Definition 2 the α-Relaxed Chord, respectively.

Our idea is to relax the requirement on full correctness of fingers and successors while still keeping the basic functionality of the Chord. Specifically, lookups will still take $O(\lg N)$ time with high probability.

Theorem 5: A lookup in a relaxed Chord X will contact no more than $O(\lg N)$ nodes with high probability, where N denotes the total number of nodes in X.

Proof. The normal key lookup algorithm for Chord in [1] can be applied here. Suppose that the node x initiates a lookup intended for the successor of a target ID id. Let p denote the node that immediately precedes id.

We will now analyze the number of query steps required to reach p. With greedy routing, if $x \neq p$, then x forwards its query to the closest predecessor of id in its finger table. Let i denote the number such that p is in the interval $[x+2^{i-1}, x+2^i)$. By definition, x's i-th finger f points to the successor of an identifier inside $[x + 2^{i-1}, x + 2^{i-1} + \delta_{x,i}) \subset [x + 2^{i-1}, x + 2^i)$.

There are two cases: (i) $f \in (x, p]$. In this case, x forwards query to f, and the relation $|f - p| \leq 2^{i-1} \leq |f - x|$ holds. Therefore, the distance from x to p is halved in this case. (ii) $f \notin (x, p]$. In this case, it implies $p \in [x + 2^{i-1}, x + 2^{i-1} + \delta_{x,i})$. Then x will forward the query via its $(i-1)$-th finger to node f', where $f' \geq x + 2^{i-2}$.
Since $|f' - p| \leq (x + 2^{i-1} + \delta_{x,i}) - (x + 2^{i-2}) = 2^{i-2} + \delta_{x,i}$, and $|x - p| \geq 2^{i-1}$, therefore, $\frac{|f'-p|}{|x-p|} \leq \frac{2^{i-2}+\delta_{x,i}}{2^{i-1}} \leq \frac{2^{i-2}+\alpha 2^{i-2}}{2^{i-1}} = \frac{1+\alpha}{2}$.

So, combining the two cases, the distance between the querying node and the node p is at least shortened by $\frac{1+\alpha}{2}$ in each step.

After $\lg_{\frac{2}{1+\alpha}} N$ forwarding steps, the distance between the current query node and p will be reduced to at most $\frac{2^n}{N^2}$. Thus, one more forwarding step from p will reach the target node. This completes the proof. □

Now let's examine the changes to the finger tables. Note that, under Protocol 2, the finger table for any node x in Chord X is kept intact; for any node y in Y, its fingers to the nearer neighbors are adjusted by locally searching on an associated interval, and its fingers to the further neighbors are directly copied from the existing ones. In details:

i) For peer y in Y, the size of the original finger table is m and will be enlarged to n by adding $(n - m)$ new entries. The operation on the finger table is: shift the original m

entries to the position of $(n-m)$ ranks higher. Then update the lower $(n-m)$ entries, as illustrated in Figure 2.

ii) For a peer x that is already present in X before merging, the finger table temporarily unchanged upon completing the merge.

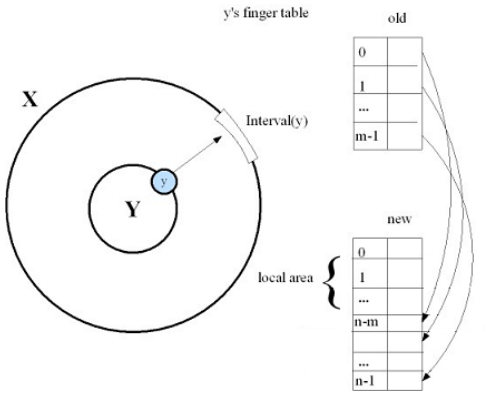

Figure 2: copying finger table

We now analyze to what extent the new Chord Z satisfies the requirement on relaxed Chord. Theorem 6 shows that the new Chord will be a Relaxed Chord with high probability under certain conditions.

Theorem 6: Suppose that Protocol 2 is applied to merge $X = Chord(N, n, id_X)$ and $Y = Chord(M, m, id_Y)$ where $m < n$, $\lg(M+N) < n$, and $n > (1+\epsilon)\lg N$, $\epsilon > 0$. If for any node $y \in Y$, $id_Y(y.finger[i]) \in id_Y(y) + 2^{i-1} + [1, \alpha 2^{i-2} - 1)$, then w.h.p. the new Chord generated by the protocol is an α-Relaxed Chord.

Proof. (Please refer to Appendix F.) □

To ensure the lookup efficiency in Relaxed Chord, we still need the stabilization of finger tables. However, it is a relatively simple job when compared with that required in the case where the merging is done by un-coordinated joining of individual nodes.

Theorem 7: For a Relaxed Chord, if each node fails with probability p, then at least $(1-p)\alpha/2$ portion of all fingers need not be updated w.h.p.

Proof. Refer to the stabilization described in Table 5. Given a finger of node x, x.finger[i], and consider an instance that violates the definition of Relaxed Chord. If x.finger[i] fails, it must be updated. Even if x.finger[i] does not fail, the new node joining between x and x.finger[i] can also trigger the update if $x.finger[i] \geq x + (1+\alpha)2^{i-2}$. Since x.finger[i] is lying in $[x+2^{i-1}, x+2^i)$ with high probability, the part of interval requiring no change is in $\alpha/2$ portion, assuming that the key of the finger is chosen from the interval randomly and

```
//periodically run
procedure x.stabilizeFingerTable
input: global parameter α, 0 < α < 1
  for i = 1 to n do
    if finger[i] failed OR finger[i] ∉ x + 2^{i-1} + [0, α2^{i-2}) do
        key = random (x + 2^{i-1} + [0, α2^{i-2}) \ finger[i]) ;
        finger[i] = lookup(key);
    endif
  endfor
end
```

Table 5: stabilization in relaxed Chord

uniformly. The probability that the finger needs no update is $(1-p)\alpha/2$. This completes the proof. □

In the original Chord protocol, each finger must be updated periodically without exception. Our modification (relaxed Chord with $\alpha < 1$) can tolerate up to $\alpha/2 < 50\%$ inexact fingers. This can substantially reduce the topology-maintenance messages and speed up the stabilization process. Relaxed Chord is especially useful for denser network because with higher probability each finger can point to an active node lying in the specified interval, and new node joining is unlikely to trigger finger updates of the other nodes.

Since each finger has multiple choices from a specified interval, to improve the robustness of the relaxed Chord, Chord can keep c node pointers for each finger if all these pointers satisfy the requirement in the definition of relaxed Chord. Therefore these c pointers in the same interval may be regarded as backup fingers for each other.

Theorem 8: Suppose that each node in Chord fails with probability p. If each finger in Relaxed Chord has $\epsilon \lg N$ backups in the same interval, where $\epsilon > -1/\lg p$, then the lookup can succeed within $O(\lg N)$ hops w.h.p.

Proof. Since each finger has $\epsilon \lg N$ backups, the probability that they all fail is $p^{\epsilon \lg N} = 1/N^{-\epsilon \lg p} < 1/N$. Therefore with probability at least $1 - 1/N$ the query can be forwarded to an active finger, shortening the distance recursively. This completes the proof. □

3.3 Doubling the size of a Chord

Sometimes the total number of the resulting networks requires the ID space to be enlarged. Fig. 3 shows a simple method for doubling the Chord ID space without communications overhead. The only work to do is add one bit to each node's ID and move the entries locally. The pointer stored in finger[i] is copied to finger[$i+1$], $1 < i \leq n$ and finger[1] stays intact.

The finger changes are performed locally for each node. The latency is mainly caused by the

broadcast to initiate the adjustment, which is $O(\lg N)$ because we ignore the time for local copying. The total number of messages is $O(N)$.

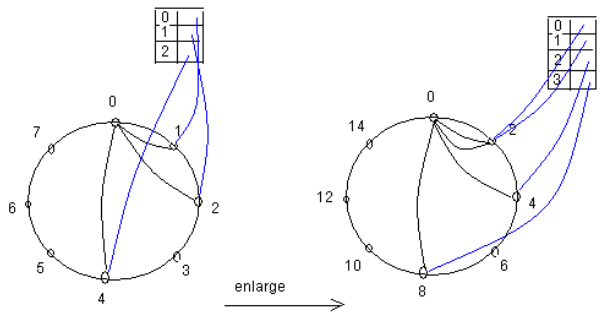

Figure 3: Algorithm 3: doubling the size of a Chord

3.4 Overall latency and message complexity

In the previous sections we have presented two different methods to merge two given Chords: one is to "concatenate" two Chords (Protocol 1), and another is to "disperse" the small Chord into the big Chord (Protocol 2). The protocols have been analyzed in terms of the number of changes in the fingers. Here we will analyze the overall latency and message complexity of the protocols. The latency will include only the time steps required for passing messages while ignoring the time for node's local processing like finger copying. Table 6 summarizes the results for these metrics, including also the number of finger changes, number of messages and overall latency required. Table 6 also includes the method for doubling the size of a Chord.

Claim 9 For Protocols 1 and 2, the number of fingers changed, the overall latency, and the total number of messages are as listed in Table 6.

Proof. (Protocol 1) The number of finger changes is given by Theorem 1, i.e. at most $2(M+N) - \lg(N \cdot M) - 4 + 2 \cdot min(N, M) = O(M+N)$. Since the merging is done by Chord X and Y separately, the overall latency is the maximum of the two latencies for X and Y. Chord X needs to update at most $2N - \lg N - 2 + min(N, M)$ fingers and each finger needs at most $\lg N$ hops to be updated. Therefore the total number of messages for merging is $(2N - \lg N - 2 + min(N, M)) \cdot \lg N + (2M - \lg M - 2 + min(N, M)) \cdot \lg M = O(N \lg N + M \lg M)$. The merging solicitation message reaches the last node in X from node x_{src} in N hops, and the last node needs at most $(\lg(M+N)) \cdot (\lg(M+N))$ hops to update its all fingers. So the latency for X is $N + \lg^2(M+N)$. The overall latency therefore is $max(N, M) + \lg^2(M+N) = O(M+N)$.

(Protocol 2) With reference to Theorem 6, the finger changes mainly happen in Chord Y. For each node y in Y, it need only modify its $(n-m)$ (or $\lg((M+N)/M)$ by simple

substitution) fingers. Therefore, the number of finger changes is given by $M \cdot (\lg((M + N)/M))$. Protocol 2 allows each y to take the joining operation concurrently upon receiving the merging broadcast. So the overall latency is $(\lg((M + N)/M)) \cdot \lg(M + N)$. The total number of messages includes two parts: the M broadcast messages, and up to $M \cdot (\lg((M + N)/M)) \cdot \lg(M + N)$ messages for updating fingers. □

Merging strategy	Number of fingers changed	Overall latency	Messages
For two Chords with $n = m$	$O(M + N)$	$O(M + N)$	$O(N \lg N + M \lg M)$
For two Chords with $n > m$	$O(M * \lg \frac{M+N}{M})$	$O((\lg \frac{M+N}{M}) \cdot \lg(M + N))$	$O(M * (\lg \frac{M+N}{M}) \cdot \lg(M + N) + M)$

Table 6: Performance comparison on two merging strategies

4 DISCUSSION AND CONCLUSION

We have presented the main steps for merging two overlays into one. Because the two overlays are reused during merging, the whole process has been substantially sped up compared with un-coordinated merging by individual nodes. Specifically for Chord, we also show that Chord can tolerate more node dynamism when it does not require accurate definition of finger table, i.e. a relaxed Chord, while still keeping the $O(\lg N)$ lookup efficiency. This kind of structure feature is very useful during P2P network evolutions.

Without a group protocol coordinating the transformation of topology, the simultaneous joining/leaving of a massive number of nodes can cause a large number of messages. Moreover, the leave-and-join can impair the routing information in the chaos and compromise the system functionality.

For future research we suggest to study the peer collaboration during the splitting and merging of various types of overlay topologies. We will analyze the evolution of network topology and its fault tolerance during node join and leaving activities in subgroups. Moreover, schemes for relocating the data on each node during merging is also an important topic.

References

[1] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balarikishnan, *Chord: A scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM, San Deigo, USA, pp. 149-160, August 2001

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content-Addressable Network*, ACM SIGCOMM, pp. 161-172, 2001.

[3] F. Kaashoek and D Karger, *Koorde: A Simple Degree-optimal Distributed Hash Table*, In 2nd International Workshop on Peer to Peer Systems, LNCS Hot Topics, Berkeley, CA, January 2003. Springer.

[4] A. Rowstron and P. Druschel, *Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-peer Systems*, IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, pages 329-350, 2001.

[5] M. Naor and U. Wieder, *Novel Architectures for P2P Applications: the Continuous-Discrete Approach*, in Proc. of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2003), San Diego, USA, pp. 50-59, June 2003.

[6] D. Lewin, *Consistent hashing and random trees: Algorithms for Caching in Distributed Networks*, Master thesis, Department of EECS, MIT, 1998.

[7] X. Wang, Y. Zhang, X. Li, and D. Loguinov, *On Zone-Balancing of Peer-to-Peer Networks: Analysis of Random Node Join*, SIGMETRICS/Performance'04, New York, pp. 211-222, June 2004.

[8] L. Devroye, *Law of the Iterated Logarithm for Order Statistics of Uniform Spacings*, Annals of Probability, vol.9, no. 5, pp. 860-867, 1981.

[9] J. Xu, A. Kumar, and X. Yu, *On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-peer Networks*, IEEE Journal on Selected Areas in Communications, Vol. 22, NO.1, pp. 151-163, January 2004.

[10] G. Manku, *Routing Networks for Distributed Hash Tables*, PODC'03, Boston, pp. 133-142, July 2003.

[11] D. Liben-Nowell, H. Balakrishnan and D. Karger, *Analysis of the Evolution of Peer-to-peer Systems*, PODC'02, pp. 233-242, July 2002.

[12] D. Karger and M. Ruhl, *Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks*, in International Workshop on Peer-to-Peer Systems (IPTPS '04) San Diego, pp. 288-297, February 2004.

[13] V. Sunderam, J. Pascoe and R. Loader, *Towards a Framework for Collaborative Peer Groups*, in Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID.03), pp. 428-433, 2003

Appendix A

Proof. As shown in Fig. 1, all the nodes in Y are to be reassigned new IDs between 2^n and $2^{n+1}-1$. Let x denote a node in Chord X. In the new Chord, x needs a new finger to point at a node y, such that $id_Z(y) = successor(x+2^n)$. Since the original ID of y (in Y) is $id_Z(y)-2^n$, it implies that y is the first node in Y that is no less than $(id_X(x) + 2^n) - 2^n = id_X(x)$.

As shown in Table 1, x_{src} finds y via a contact node y_{src} in Y. Then it sets its last finger to point at y and sends this information to its successor. So when each x receives the merging message $< x_{src}, y, "merge" >$ from its predecessor, y is exactly the successor of $id_X(x.predecessor)$ in Chord Y. So the following relation must be satisfied: $id_X(x.predecessor) \leq id_Y(y) \leq id_X(x)$ when the lookup is executed, which proves item (2) of the results. For Chord X, there are N nodes, each of which chooses one node from Y as its last finger. Therefore the total number of different last fingers for nodes in X is no more than $min(N, M)$. A similar result can be obtained for Chord Y, and hence a total of $2 \cdot min(M, N)$ for all the nodes in both X and Y. This completes the proof. □

Appendix B

Proof. We recall a result in [7] (Theorem 1 specifically) which implies that
$|id_X(x) - id_X(x.predecessor)| \leq (1+\epsilon)\lg(N) * 2^n/N$
with probability $1 - N^{-\epsilon}$. Notice that the distance between the node handling the query and the destination is halved in each step, and it is initially equal to $|id_X(x) - id_X(x.predecessor)|$, so the total number of contacted nodes in Chord Y is $\lg|id_X(x) - id_X(x.predecessor)| \leq (n - \lg\frac{N}{\lg N} + \lg(1+\epsilon))$. □

Appendix C

Proof. Lemma 1 gives the number of newly created fingers.

Now consider a node x in Chord X during the concatenation (see Fig. 1 for illustration). It should be easy to see that if x has a larger value of ID (and hence is near the range of Y), then more of its finger entries will have to be redirected (to point at nodes in Y).

Note that, when Chord Y's address space is fully occupied (i.e. $M = 2^m$), the modified pointers in the finger table all have different values. Also, when Chord X is fully occupied (i.e. $N = 2^n$), the number of affected nodes is the largest and the number of affected fingers is also the largest. So, to derive an upper bound, we assume that $N = M = 2^n$. For convenience, we write x_i to denote a node with ID $= i$.

The nodes in the range of $[0, 2^{n-1})$, i.e. $x_0, x_1, \cdots, x_{2^{n-1}-1}$, will modify 0 finger items;
The nodes in $[2^{n-1}, 2^{n-1} + 2^{n-2})$, i.e., $x_{2^{n-1}}, x_{2^{n-1}+1}, \cdots, x_{2^{n-1}+2^{n-2}-1}$, will each modify 1 finger item (the last entry in old finger table);
.....
The nodes in $[2^{n-1} + 2^{n-2} + \cdots + 2^{n-k+1}, 2^{n-1} + 2^{n-2} + \cdots + 2^{n-k+1} + 2^{n-k})$ will each modify k finger items (the last k entries in the old finger table); In total $k \cdot 2^{n-k}$ entries will be affected.
.....
The node x_{2^n-1} will modify $(n-1)$ fingers.

Therefore, the total number of fingers S_n that need to be modified for the nodes in X is given by:

$$S_n = \sum_{k=0}^{n} k \cdot 2^{n-k} = 2^{n+1} - n - 2.$$

It should not be difficult to see from the above that if n is substituted by $\lg N$, the argument still holds true. The scenario is similar with nodes in Y. Therefore the maximum number of modified fingers is $2(M+N) - \lg(N \cdot M) - 4$. This completes the proof. □

Appendix D

Proof. By assumption, each node x in Chord X is assigned an ID by choosing a point independently, uniformly and randomly in the ID space. Therefore, the probability that x is assigned an ID=i is given by $p = \frac{1}{2^n}$. In notation, $Pr(id_X(x) = i) = p, \forall x \in X$. In the rest of the proof, x is used to denote $id_X(x)$ as well.

Write the points belonging to Interval(y) as $\{i_0, i_1, ..., i_{\pi-1}\}$. Following the merging process in Table 2, we have:
p_0:=Pr(y is inserted to point i_0) = Pr(point i_0 is empty, unoccupied by any of the N nodes of X)= $(1-p)^N$;
p_1:=Pr(y is inserted to point i_1) = Pr(!empty(i_0) and empty(i_1))
= Pr($\exists x_j \in$ X, $x_j = i_0$ and $\forall x_i \in$ X, $x_i \neq i_1$)
$= \sum_{j=1}^{N} \Pr(x_j = i_0) \cdot \Pr(\forall x_i \in X, x_i \neq i_1 \; and \; x_j = i_0)$
$= \sum_{j=1}^{N} \Pr(x_j = i_0) \cdot \prod_{i \neq j, i=1}^{N} \Pr\,(x_i \neq i_1)$
$= Np(1-p)^{N-1}$;

p_k:= Pr(y is inserted to point i_k)
= Pr($\exists\{x_0, x_1, ..., x_{k-1}\} \subseteq X,\; x_0 = i_0, x_1 = i_1, ..., x_{k-1} = i_{k-1};\; and\; \forall x_k \in X,\; x_k \neq i_k$)
$$= \begin{cases} \binom{N}{k} p^k (1-p)^{N-k} & , if \quad k \leq N \\ 0, \quad if \quad k > N \end{cases}$$

So, by summing up:
Expectation(The offset of y's ID from the point i_0) $= \sum_{k=0}^{N} k p_k$
$= \sum_{k=0}^{N} k \binom{N}{k} p^k (1-p)^{N-k}$, ($\because \binom{N}{k} < N^k$)
$< \sum_{k=0}^{N} k (\frac{Np}{1-p})^k (1-p)^N = (\frac{1}{1+\beta/N})^N \sum_{k=0}^{N} k\beta^k$
$\leq \frac{1}{1+\beta} * \frac{\beta}{(1-\beta)^2}$, $(\because \sum_{i=1}^{N} i\beta^i = \frac{N\beta^{N+2} + \beta - (N+1)\beta^{N+1}}{(1-\beta)^2} < \frac{\beta}{(1-\beta)^2}$, if $0 < \beta < 1)$.

Furthermore,
Pr(y is inserted to Interval(y)) $= \sum_{k=0}^{\min(\pi, N)} \binom{N}{k} p^k (1-p)^{N-k}$

Pr(y is inserted to any of the first c points of Interval(y)) $= \sum_{k=0}^{c} p_k = 1 - \sum_{k=1}^{N-c} p_{k+c} =$
$1 - \sum_{k=1}^{N-c} \binom{N}{c+k} p^k (1-p)^{N-k} (\frac{p}{1-p})^c$

$> 1 - \sum_{k=1}^{N-c} \binom{N}{k} p^k (1-p)^{N-k} (\frac{Np}{1-p})^c$

$> 1 - (\frac{Np}{1-p})^c = 1 - \beta^c$, since $\binom{N}{c+k} = \binom{N}{k} \frac{(N-k)(N-k-1)\cdots(N-k-c+1)}{(k+c)(k+c-1)\cdots(k+1)} < \binom{N}{k} N^c$

This completes the proof. □

Appendix E

Proof. Let S_i denote the gap between x_i and its successor along the Chord ring X, where the size of the overall address space is 2^n. Normalize the set $\{S_i\}$ by dividing each S_i by 2^n. The set of $(S_1, S_2, ..., S_N)$ is uniformly distributed on the simplex $\{(w_1, w_2, ..., w_N) : w_i \geq 0, \sum_{i=1}^{N} w_i = 1\}$ and accordingly [8], we have

$$Pr(S_1 > a, S_2 > a, \ldots, S_N > a) = \begin{cases} (1-aN)^{N-1}, & aN < 1 \\ 0, & aN \geq 1 \end{cases}$$

Let $a = \frac{1}{2^n}$, then the probability above is given by $(1 - \frac{N}{2^n})^{N-1}$
$\geq 1 - \frac{N(N-1)}{2^n}$
$\geq 1 - \frac{N^2}{2^n}$
$\geq 1 - \frac{N^2}{N^{2+\epsilon}}$
$= 1 - \frac{1}{N^\epsilon}$

which means the probability that any (normalized) gap is greater than $\frac{1}{2^n}$ is at least $1 - N^{-\epsilon}$.

Therefore, each $Interval(y)$ contains at most $\frac{2^{n-m}/2^n}{a} = 2^{n-m}$ nodes with high probability. In other words, with high probability, there is still an unused ID inside $Interval(y)$ for a given y to take up. This completes the proof. □

Appendix F

Proof. By assumption, Chord X is already an exact Chord. For any $x.finger[i]$, it can only lie in one interval with size 2^{n-m}. Rarely would the merging operation destroy x's condition for Relaxed Chord. Therefore we consider the node y in Y. Denote $f_i = y.finger[i]$. By assumption, we have
$id_Y(f_i) = id_Y(y) + 2^{i-1} + c$, where $c \in [1, \alpha 2^{i-2} - 1)$
or $id_Y(f_i) \cdot 2^{n-m} = id_Y(y) \cdot 2^{n-m} + 2^{n-m+i-1} + c \cdot 2^{n-m}$
Theorem 2 assures that any y can be inserted to its intervals with high probability. Furthermore, it implies that w.h.p. $id_Z(y)$ and $id_Y(y) \cdot 2^{n-m}$ differ by at most 2^{n-m}. Thus we have:

$id_Z(f_i) = id_Z(y) + 2^{n-m+i-1} + c \cdot 2^{n-m} + c^{'}$, where $c^{'}$ is a constant in $[-2^{n-m}, 2^{n-m}]$.

In the new Chord Z, f_i is copied directly as the $(n-m+i)$-th finger of the node y. Therefore, to satisfy the requirement of a Relaxed Chord, it is only necessary to satisfy:

$0 \leq c \cdot 2^{n-m} + c^{'} < \alpha 2^{n-m+i-2}$, or,
$0 \leq c + \frac{c^{'}}{2^{n-m}} < \alpha 2^{i-2}$

The inequalities hold because $c \in [1, \alpha 2^{i-2} - 1)$ and $\frac{c^{'}}{2^{n-m}} \in [-1, 1]$. This completes the proof that the new Chord is a relaxed Chord with high probability. $\square$