

# Merging ring structured overlay indices: A data-centric approach

Anwitaman Datta, NTU Singapore  
anwitaman@ntu.edu.sg

**Abstract.** Peer-to-Peer index structures distributed and managed over the planet, commonly known as structured overlays (e.g., Distributed Hash Tables) have been touted to play the role of a fundamental building block for internet-scale distributed systems. Traditional designs consider incremental or possibly even parallelized construction of a single overlay, which implicitly assumes global control and coordination to enforce the construction of an unique overlay. However, if merger of originally isolated overlays is made possible, then one can realize decentralized bootstrapping of overlays. So to say, smaller overlays can be constructed using any of the traditional mechanisms, which can then organically coalesce to form a larger overlay. Such a self-organizational attribute of decentralized bootstrapping is of paramount importance for large scale systems. In our previous works, we explained the challenges of merging important families of (tree and ring) structured overlays [6], and identified that tree structured overlays are relatively easier to merge in a transparent manner [5]. In this paper we investigate how two ring structured overlays can be merged, both in terms of the necessary algorithms, as well as how it performs during the merger process. We also introduce interesting new metrics to evaluate the merger process and carry out asymptotic analysis for estimating the same, besides conducting simulation experiments to validate the theory as well as measure other aspects of the overlays' performance.

**Keywords:** structured overlay, index merger, decentralized bootstrapping, network data independence

## 1 Introduction

Since the beginning of this decade, numerous structured overlays (e.g., distributed hash tables) have been proposed, and some of them have been prototyped and deployed typically in controlled environments. Structured overlays are touted as a basic building block for planetary scale distributed systems and applications by providing both routing primitives as well as by acting as a distributed index supporting decentralized search.

The operational phase of such structured overlays is often decentralized (including self-maintenance). Construction or bootstrapping of overlays has traditionally assumed a quasi-sequential approach, in which new peers would join by contacting some existing members of the network [7, 21, 23]. More recently, parallelized structured overlay construction mechanisms have been developed [1, 19], again assuming participating peers already know and can contact each other, e.g., using an unstructured connected network. All these structured overlay bootstrapping strategies thus implicitly assume logical centralization - even if distributed, e.g., a set of globally known rendezvous "seed" peers - which facilitates new peers to join and form one single network.

Imagine that any or several of these existing techniques are used to construct individual overlays based on distinct (disjoint) set of *seed* peers. These overlays will operate as isolated islands. If at a later time, some peers from one overlay network meets peers from another network (by whichever mechanism), existing overlay maintenance and construction mechanisms will fail to deal with it [6]. Facilitating merger of such originally isolated but evolved overlays into a single overlay network paves the way for decentralized bootstrapping of overlay networks. People can set up their own bootstrap seed peers to build smaller but functional overlay networks, and when they meet and wish to merge their network with others, the isolated overlays can coalesce to grow organically into a single larger overlay. This is analogous to the problem of merging index in data

management [4, 16], and it is desirable to achieve *network data independence* [12], which in this context is that such changes don't disrupt upper layer functionalities. Generally speaking, decentralized bootstrapping is a desirable self-organizational property for large-scale systems. Merger and thus decentralized bootstrapping occurs trivially in unstructured P2P systems, while structured overlays lack it. Merging tree-structured P-Grid [1] indices was speculated to be relatively easier than to merge ring overlay indices [6]. We devised mechanisms to do so in our previous work [5].

In this paper, we study how two originally isolated ring-structured overlays like Chord [23] can be merged, and how the system performs during such merger process.<sup>1</sup> Chord is one of the earliest proposed structured overlays, and owing to the simplicity of its ring based topology, as well as its robustness [10], is predominantly used and studied. Many variations, optimizing the use of routing tables or the self-stabilization algorithms [2, 9, 17, 18] have also been derived. Given its predominance, it is imperative to design mechanisms to merge ring structured overlays.

In fact, both Chord's original self-stabilization mechanism as well as several other follow up works, e.g., [3, 8, 22, 15] would appear to achieve ring merger but they do so in a limited sense. Specifically, these do not consider the key-management aspect of structured overlays at all, but restricts its focus to only end-to-end routing between peers over the overlay. While for network centric applications routing correctness is adequate, data-centric applications also need to manage the binding of keys to correct peers. One can argue that once the routing network is correctly merged, data can be shipped to appropriate peers afterwards. However, such an approach leads to disruption of the overlay's functionality until not only the routing network merger process is completed, but also the background data shipping is achieved. It is thus essential to take into account the data shipping mechanism ensuring that correct key-peer bindings are established explicitly and as soon as possible as part of the merger process.

Algorithmically the proposed ring merger mechanisms are intuitive, simple and similar in spirit to the simplicity of ring structured overlays themselves and can be readily integrated to Chord's self-stabilization algorithms. However as speculated in our previous work [6], ring-structured overlays are indeed more susceptible to disruption than tree-structured overlays during merger, though the performance of rings under merger is still reasonably good. Besides measuring performance of merger using metrics like *recall* introduced in our previous work on tree structured overlay merger, we also identify new metrics of interest to judge the performance of the merger algorithms themselves, namely the efficiency of routing network merger based on peers' successor changes as well as efficiency of data transfer. Such evaluation of the merger process of ring based overlays - how the overlay performs during merger and how the merger algorithm itself performs - have not been studied in related works, e.g. [8, 22] which deal with ring merger in a limited scope, where most evaluation is carried out for the resulting post merger network. Identification of these new metrics of interest and carrying out the corresponding evaluations are also novelties of the presented work.

## 2 Related work

In our previous work we have addressed the problem of merger of two tree structured overlays [5]. However, the problem of merging two ring structured overlays is structurally different, and has its own distinct challenges [6] that we try to address here. The networking community has looked

---

<sup>1</sup> Both due to popularity and familiarity of Chord in distributed systems community as well lack of space, we skip description of the Chord network.

into a similar problem of dealing with multiple overlays based on potentially diverse topologies and protocols. The objective has typically been of end-to-end communication of hosts. Some of these [14, 3] have borrowed ideas from the networking domain - managing individual overlays as autonomous systems (AS)/domains and routing traffic from one overlay to another using BGP like mechanisms based on certain peers specifically designated as gateways. In the discussion of [14] the authors themselves point out the limitation of their approach in dealing with application layer functionalities, e.g., storage. There are several other efforts which try to integrate multiple overlays based on the idea of domains [8, 11], and have the same limitations in that routing to a specific end node can be achieved correctly but that is not sufficient to locate the right keys.

In a flat (domainless) overlay, one needs to know only the name of the resource to be sought, and locate it. Such a data-structure can thus readily be used as an index. However if it is necessary to already know specific domain, then it can not be used as an universal index as has been the case with flat domainless overlays. Such domainless flat data-structures was also one of the original appeals of universal applicability of first generation structured overlays like distributed hash tables (DHTs) [23, 20, 21].

Furthermore none of these approaches evaluate either the cost of merger or the performance of the overlay during merger, but only the properties of the eventually merged network. In contrast we study specifically the dynamics of the merger process itself.

Association of keys with peers can potentially change with merger of overlays. Communication oriented applications do not need to deal with management of the stored content, which is however a critical issue for any data-oriented application, particularly if full *recall* (=1) is desired or needed. This issue has been overlooked in previous works [3, 8, 22] which propose ring merger to achieve routing correctness.

A gossip based bootstrapping service [13] has been proposed as a generic mechanism for constructing an overlay in a parallelized manner [19] from scratch. We speculate that similar gossip based approach can also be used to merge multiple overlay based routing networks. If such an approach is however indeed used, again, in itself it will not deal with the key-to-peer associations, and thus will not be directly useful. This is similar to *re-build* of an index[16].

Merger of overlays shares some resemblance with another essentially different issue - churn or membership dynamics. Merger of two overlays, seen from the perspective of individual peers (which is how these decentralized systems operate) can look very similar to new peers joining the network.

However, churn is a gradual process, and the system needs to continuously perform repair operations to rectify local view of peers in order to deal with the continuous membership changes. Churn thus has analogous effect as *in-place* maintenance in an index[16]. When two isolated networks need to merge, the magnitude of the population change with respect to their original population size is very high.

Many network-centric applications however need only the correctness of routing, and thus most overlay related work in networking community focuses primarily in maintaining the topology invariant to guarantee routing correctness.

### 3 Merging rings

**Definition 1.** *A structured overlay based index needs to meet two objectives to guarantee functional correctness in locating stored key-value pairs:*

(i) *Correctness of routing*: Starting from any peer, it should be possible to reach the correct peer(s) which are responsible for a specific resource (key).

(ii) *Correctness and completeness of keys-to-peers binding*: Any and all peers responsible for a particular key-space partition should have all the corresponding keys (and ideally, none other).

Consider two originally isolated Chord networks  $\mathcal{N}_1$  and  $\mathcal{N}_2$  as shown in Figure 1. Peers in the network are marked as nodes on the ring identifier space. As long as they stay isolated, each stand-alone network operates perfectly. However if merger of the networks is partial it will lead to inconsistencies. If either one of the two clauses of Definition 1 are violated, then due to various reasons there will be disruption in the overlay's usability in typical data-centric applications.

Consider the following scenario from Figure 1. Key-value pairs corresponding to the keys '11' and '9' exist originally in the network  $\mathcal{N}_1$ . Peer 12 will be responsible for these keys, and all queries related to them need to be routed to peer 12. Likewise, key '7' exists in network  $\mathcal{N}_2$  and peer 10 is responsible for it.

When peer 5 from network  $\mathcal{N}_2$  and peer 6 from network  $\mathcal{N}_1$  come in contact with each other, peer 5 will identify peer 6 as its new legitimate successor (instead of peer 10). Until peer 8 updates its own successor to be peer 10, a query at peer 8 for key 9 will be forwarded to peer 12 instead of peer 10, and hence the key won't be found. Likewise, if a query for the key 7 arrives subsequently at peer 5, it will be forwarded to peer 6. Peer 6 will 'rightly' forward the query to peer 8. Unless the key 7, which was originally stored in peer 10 has in the meanwhile been moved to peer 8 which is its legitimate placement in the merged network, key 7 will become inaccessible to applications. Note that this key was previously available to all those applications which were relying on network  $\mathcal{N}_2$ , and from the end-user's perspective, such disruption because of underlying layer's dynamics is undesirable. The merger process should be transparent. The former anomaly is because of wrong routing - a query for key 9 should be routed to peer 10 in a merged network, while the later is because of wrong peer-to-key binding - the query for key 7 should be and is routed to peer 8 in a merged network, but peer 8 does not have the key until the correct key to peer binding is reestablished.

Also, if peer 6 was an isolated peer joining network  $\mathcal{N}_2$ , it would suffice for peer 5 to assign 6 as its successor, and peer 6 assigning peer 10 as its successor. However, when peer 6 is not an isolated peer, but instead itself belongs to another structured overlay, it will need to ascertain which peer should be its successor. In this example, peer 6 can retain its original successor, namely peer 8. However, if peer 8 is not informed about the merger operation (which it has no way to know about by itself from interactions between peers 5 and 6), peer 8 will continue to consider peer 12 as its successor, instead of peer 10. This will lead the network to become loopy, disrupting correct routing. This highlights the need to propagate the ring merger to other peers over both the rings.

Another observation we will like to make here is that over the ring identifier space, there is a strict ordering among the peers, and hence, there is a unique merged network in terms of choice of successor/predecessor nodes at each peer, and hence, the ring merger process naturally can be conducted in a parallelized manner over disjoint parts of the identifier space. Since all operations - verification and change of successor, key shipment, etc are local interaction among peers, such parallelization is easy and increases significantly the speed of the merger process. This also allows natural extension to deal with simultaneous merger of multiple rings.

If same key exists in both networks, how the keys/values are merged depends on application semantics, and can not be handled at the overlay layer. Such issues still make the merger process

---

**Algorithm 1** *u.Interact(v)*

---

```
{Peer u initiates interaction with another peer v.}
v* := v.Findkey(u) {Locate peer responsible for key 'u' by querying at v to determine suitable interaction partner in v's network for u.}
if v* ≠ v {Peer u will initiate a new interaction with peer v* (at next time round) since it 'seems' to be more suitable interaction partner according to v.} then
  u.Interact(v*);
else
  {u and v are in each other's neighborhood on the ring identifier space.}
  x' := u.Successor;
  if v ∈ (u, x') {Peer v should replace x' as peer u's successor.} then
    u.Successor := v;
    ∀k ∈ v.LocallyStoredKeys ∧ k ∉ (u, v] Move key k to peer u;
    ∀k ∈ x'.LocallyStoredKeys ∧ k ∉ (v, x'] Move key k to peer v; {Moving keys from x' to v in this step is optional - not/used in some experiments.}
    v.Interact(x');
    Optional step: Initiate interactions among random peers in u's routing table with random peers in v's routing table;
    {Used to accelerate the merger process by spreading the merger interactions in a manner similar to broadcast/multicast on structured overlays.}
  end if
end if
```

---

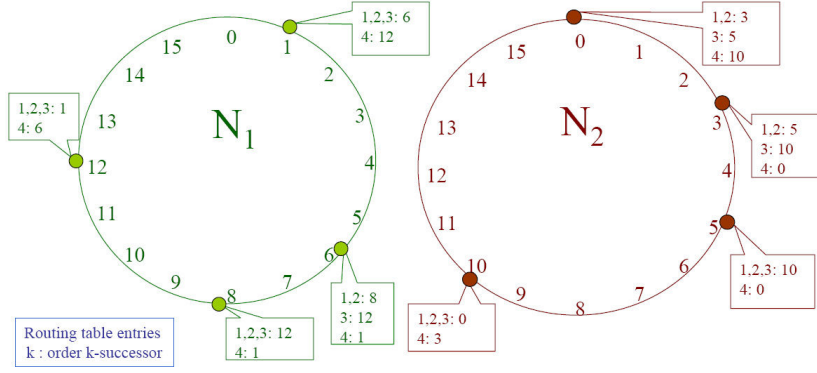
not so transparent for the applications and may need end user intervention. So the way we interpret transparency is as follows. All key-value pairs which were originally accessible to the applications (and end users) before the merger process started continue to stay accessible, which is ensured if both routing is correct and key to peer binding is correct.

### 3.1 Reestablishing ring invariant

From the local perspective of individual peers there is not much difference between a churn event and a merger related change. The basic ideas necessary to merge two ring structured routing networks are intuitive and use a relatively proactive variation of Chord's original self-stabilization. Algorithm 1 describes a single step of the merger operations, which is also illustrated in Figure 2. If interacting peers change only their local information then the network will become loopy. It is essential that the merger process is propagated to other peers in the network. This can be done sequentially along the rings, but the merger process can also be accelerated by parallelizing the interactions by inducing more interactions using peers' routing table entries (*optional step* in the algorithm).

**Estimating overheads** From the perspective of any peer in  $\mathcal{N}_1$ , the successor will change if at least one out of the  $N_2$  peers from  $\mathcal{N}_2$  have identifier within the next  $1/N_1$  stretch of the key-space for which its present successor is responsible on an average. Any particular peer from  $\mathcal{N}_2$  has an identifier for this stretch with probability  $1/N_1$ . The number of peers in this stretch is thus distributed as  $Binomial(N_2, 1/N_1)$ , which approaches a Poisson distribution with expectation  $N_2/N_1$  as  $N_2 \rightarrow \infty$ . Hence, a peer from  $\mathcal{N}_1$  will have its successor unchanged with probability  $e^{-\lambda_1}$  where  $\lambda_1 = N_2/N_1$ . Thus each of the  $N_1$  peers will have their successor node changed with a probability  $1 - e^{-\lambda_1}$  i.i.d. Peers in  $\mathcal{N}_2$  will be affected with a parameter  $\lambda_2 = N_1/N_2$  (symmetry).

The number of peers whose successor will change in  $\mathcal{N}_i$  is thus distributed as  $Binomial(N_i, 1 - e^{-\lambda_i})$  for  $i = 1, 2$ . Hence the expected number of pairs which will need to correct their successor



**Fig. 1.** Ring-structured Chord overlays

and predecessor is approximately  $N_1(1 - e^{-\lambda_1}) + N_2(1 - e^{-\lambda_2})$ . If  $N_2 \gg N_1$  then by simplification of the exponential expansion we obtain the expected number of successor changes to be  $\sim 2N_1$ . More interestingly, if the two networks are of roughly equal size, then  $1 - e^{-1}$  fraction (approx. 63.2%) of all the peers are expected to have changes in their successors and predecessors.

The above expression provides the asymptotic estimate of the expected minimum number of changes in successors required to merge two networks. We observe from the expressions that the smaller of the two networks determines the cost of merging the two networks. The convergence to Poisson distribution happens only asymptotically, and hence is a rough and in fact pessimistic estimate as can be observed from experiment result in Figure 3(b).

The basic idea of Algorithm 1 for reestablishing a merged ring is that when two suitable peers from different networks meet, they replace each others' successor and predecessor (immediate neighbor), and then this information is communicated to the original immediate neighbors, and the merger process continues.

If the ring merger is percolated sequentially along the ring (without the optional step of Algorithm 1) the latency of such a process started because of two peers from the two networks will be  $O(N_1 + N_2)$  - this is the time required to percolate the information that the ring neighborhood has changed and to discover the correct neighbor when peers from both the original networks are considered together. The propagation of the merger process can however be accelerated. Because of the property of an unique ordering of the peers on the identifier ring, parallelization is straightforward. The optional step in the algorithm does exactly that by propagating the merger process along peers' routing entries, and the merger process spreads exponentially so that the merged routing ring is established with a latency of  $O(\text{Min}(\log_2 N_1, \log_2 N_2))$ .

Every peer  $x$  checks if  $\exists u, w \mid u \neq w \wedge u.\text{Successor} = x \wedge w.\text{Successor} = x$ , in which case interaction between peers  $u$  &  $w$  is prompted using Algorithm 1. This background ring consistency maintenance mechanism is a continuous process and compliments the merger process, and takes care of anomalies that may arise in the ring consistency. This mechanism runs continuously in the background, and only when some inconsistencies are detected does it propagate and trigger new interactions. For instance, in our example networks, if peer 5 meets peer 8, according to our algorithm it will first assign 8 as its current successor, and subsequently will discover peer 6 to be the correct successor. Generally the changes will be because of churn, and hence the change

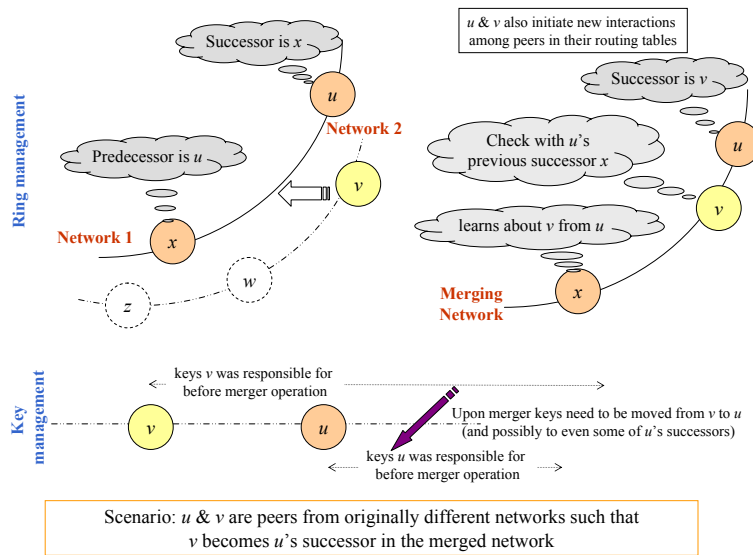


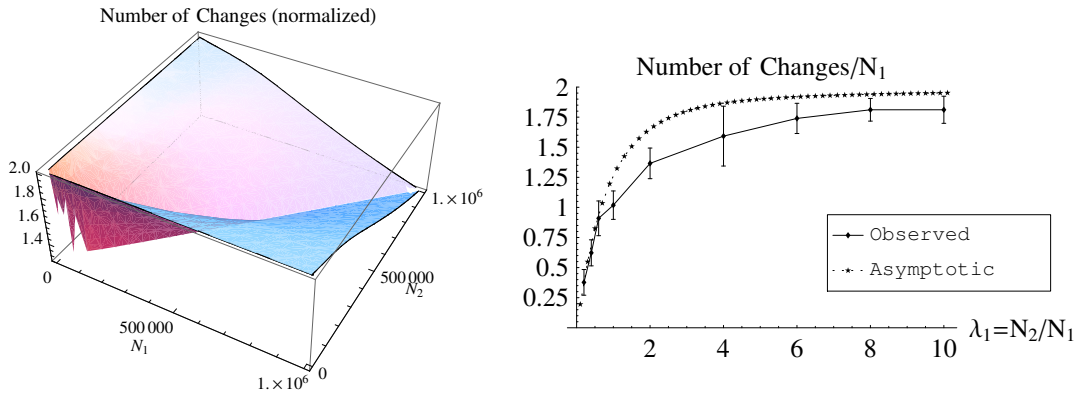
Fig. 2. Ring merger operations

operations will be localized. Note that if no more inconsistencies are found the merger algorithm again automatically fades out. Thus, there is no need for peers to locally discern between a churn event from a merger event. The essential idea is that whenever there is a ring inconsistency, more than one peer will designate a same peer as their respective successor. For instance, in our running example, both peers 5 and 6 will consider peer 8 as their successor momentarily. Hence this peer can prompt the peer with 'wrong' successor to interact with the other peer.

The precise minimum number of changes required to merge two networks will depend on the actual distribution of the peer identifiers in the two originally isolated networks. The actual number of changes may depend on both the details of implementation of the merger algorithm as well as be influenced by some extrinsic factors like which peers interact for a specific sequence of merger operation. For example, in the above example the first change - peer 5 designating peer 8 as its successor - is not essential, and a different algorithm may be able to eradicate more of such unnecessary changes. Thus the *efficiency of successor changes* which we denote as  $\eta_s$  and define as the ratio of the minimum number of successor changes that are necessary to merge the network with respect to the actual number of successor changes that happen during the merger process is an interesting evaluation metric for overlay merger algorithms.

### 3.2 Managing Keys on the Merged Ring

Establishing the ring in itself is however not sufficient in an overlay network based index. In order to really find all keys (which originally existed in at least one of the two networks) from any peer in the merged network, it will still be necessary to transfer the corresponding key/value data to the "possibly" different peer which has become responsible for the key in the merged network. To make things worse, in a ring based network the queries will be routed to the new peer which is



(a) A theoretical estimate of the number of successor/predecessor changes required, normalized by the size of the smaller network. (b) Asymptotic vs observed number of successor changes. The error bar corresponds to twice the standard deviation, and shows that the theory provides a decent upper bound.

**Fig. 3.** Expected minimum number of successor changes for merging two ring overlays

responsible for a key, so that even after the reestablishment of the ring itself, some keys that could be found in the original networks may not be immediately accessible, and will need to wait until the keys are moved to the new corresponding peer.

Lets consider that before the networks started merging, network  $\mathcal{N}_i$  had key set  $\mathcal{D}_i$  such that  $|\mathcal{D}_i| = D_i$ . Furthermore, if we consider that  $\alpha$  fraction of the keys in the two networks is exclusive, that is  $|\mathcal{D}_1 \cap \mathcal{D}_2| = \alpha|\mathcal{D}_1 \cup \mathcal{D}_2|$ , then on an average, if a  $\mathcal{N}_i$  node's successor changes, it will be necessary to transfer on an average  $\alpha$  fraction of the data from network  $\mathcal{N}_j$ 's  $\frac{1}{N_1+N_2}$  stretch of the key-space. Thus, on an average, the minimum<sup>2</sup> required transfer of unique data from members of original networks  $\mathcal{N}_j$  to  $\mathcal{N}_i$  will be  $D_{tx}^{j \rightarrow i} = N_i(1 - e^{-\lambda_i})\alpha \frac{D_j}{N_i+N_j} = (1 - e^{-\lambda_i})\alpha \frac{D_j}{1+\lambda_i}$ .

Apart from assigning the data corresponding to a key on the key-space to the peer which is the successor for that key, ring based topologies provide fault-tolerance by replicating the same data at  $f$  consecutive peers on the ring.<sup>3</sup> Given the strict choice of  $f$  as neighborhood changes, the transferred data will in-fact have to be replicated at the precise  $f$  consecutive peers of the merged network, determining the actual minimal bandwidth consumption. Similarly, some of the original  $f$  replicas will need to discard the originally replicated content. This incurs additional overheads, however, does not directly affect correctness of the overlay. Thus, for the sake of simplicity we have ignored the effects of replication in our analysis as well as evaluation.

Observe that at any time a peer should only hold data corresponding to keys for which it is considered to be the successor on the identifier ring. Thus, due to either churn (new node joining) or network merger operations, if a peer has a new predecessor, then the part of the key space it is responsible for shrinks. Corresponding keys need to be transferred to such a new predecessor. Note that transferring the key just to its immediate predecessor may be inadequate. For example consider peer 10 originally held data corresponding to key 6. After the routing network merger, peer 8 is peer 10's predecessor. Peer 10 however does not in general have the local knowledge about the existence of peer 6, who will be the legitimate owner of the key 6 in a merged network.

<sup>2</sup> The actual implementation of such a data transfer will need to identify the distinct data in the two networks and transfer only the non-intersecting one, in order to achieve this minimal effort. This is an orthogonal but important practical issue that any implementation will need to look into.

<sup>3</sup> The parameter  $f$  is a predetermined global constant determined by the system designer.

A simple strategy will be for peer 10 to transfer data to peer 8 and let peer 8 determine the best course of action. Such a data maintenance mechanism based on strictly local information is the simplest in ring structured overlays, and is what we use. This however adds to the delay in the completion of the merger process.

From the perspective of the smaller network, the number of peers from the larger network that comes between two originally consecutive peers is likely to be relatively large, and hence will dominate the data transfer delay. The number of peers that will be positioned between two consecutive peers of the smaller network follow approximately a Poisson distribution with mean  $\lambda = \text{Max}(\lambda_1, \lambda_2)$ . This distribution does not only determine the delay in completing the merger process, but also can be used to estimate data transfer efficiency asymptotically.

From data transfer efficiency perspective the best thing to do will be to transfer each key directly to its corresponding peer in the merged ring. A lazier approach than our's which waits for completion of the merging process of the routing network first, and then determining the "correct" peers for the keys, and transferring keys directly to such peers is arguably more efficient.<sup>4</sup> Unfortunately such a strategy will render the network in an inconsistent state for much longer, severely disrupting the network's usability as shown in plot  $R_{ndm}$  in figure 4(b). The sequential approach we currently use leads to extra data transfers.  $k$  new peers are in between two original peers with a probability  $P_k = \frac{\lambda^k e^{-\lambda}}{k!}$ . If there are  $k$  new peers in between, then the expected number of transfers is  $\frac{1+2+\dots+k}{k} = \frac{k+1}{2}$  assuming uniform distribution of keys (as is the case in DHTs). Thus the *data transfer efficiency* can be determined asymptotically to be  $\eta_d = \frac{1}{\sum \frac{k+1}{2} P_k} = \frac{2}{1+\lambda}$  where  $\lambda$  is the relative size of the larger network with respect to the smaller one. This expression shows that the data transfer efficiency decreases as the difference between the sizes of the two networks increase. Since most of the data is expected to be transferred from peers originally in the smaller network, the absolute amount of data itself to be transferred will be less and determined by the volume of data in the smaller network, a fact not reflected by the data transfer efficiency.

## 4 Evaluation

Our simulator supports Chord networks on an arbitrary sized ring identifier space, where the density of nodes determines the peer population, which are distributed uniformly randomly. In the below examples, unless otherwise mentioned otherwise, the default density of nodes on the identifier space was 0.1 in each original network. On a 17 bit identifier space, this corresponds to approximately 13K peers in a network. Error bars in the figures below correspond to the standard deviation from ten repetitions of the experiments.

**Performance:** We evaluate the routing network's correctness by looking at whether a routing request reaches the correct peer.  $i \rightarrow j$  in Figure 4(a) represents routing requests starting in peers originally from network  $i$  for destination peers originally from network  $j$ . Ideally, route requests between peers originally belonging to the same network should always work correctly, but as can be seen from the figure, there is temporary disruption, where up to 13% of the route requests did not reach the correct target, but such disruption is short lived, and since the routing network is merged pretty fast because the merger process is propagated exponentially, correct routing for all requests is achieved soon. Success rate of routing requests between peers originally in the two different isolated overlays also grows rapidly to one.

<sup>4</sup> It will involve overheads for determining such correct peer for each key.

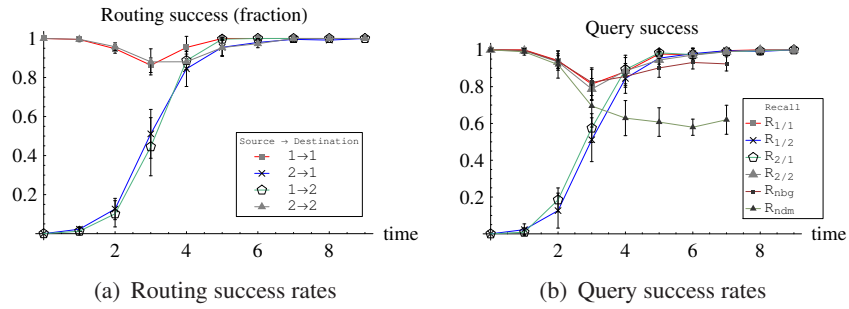


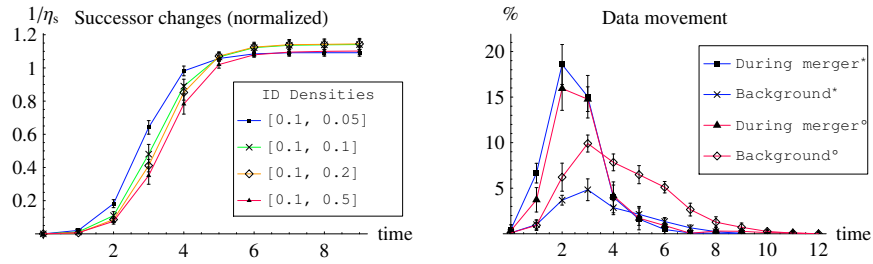
Fig. 4. Routing and query performance during merger process.

We use the information retrieval metric of *recall* to quantify the overlay’s functional correctness in terms of answering queries. It can be interpreted as the probability that a key-value pair can be located if the key-value pair is actually present in the network, by querying for the corresponding key at any peer in the network.  $R_{i/j}$  represents the recall for a key which is originally in network  $i$  while the query is issued by a peer originally in network  $j$ . From Figure 4(b) we see that up to  $\sim 28 - 30\%$  of queries could temporarily not be answered even for keys which originally belonged to the same network as the querying peer, even when we ship data during the merger process, and also carry out the background synchronization process continuously. This figure is much higher than the amount of failed routing requests, and is due to the fact that even if the query is routed to the correct peer for a corresponding key, the key itself may yet not be at the right peer until the data shipment and synchronization is completed. The good news is, using our mechanisms the network recovers rather fast, and can again answer all queries correctly. Also, queries for keys originally in the other network can all be correctly answered after the merger process. If *no background data synchronization* is carried out then approximately 10% queries can not be answered after the merger of the routing networks as seen from the plot  $R_{nbg}$ , which roughly corresponds to the volume of data transferred during the background process as shown in figure 5(b). The plot corresponding to  $R_{ndm}$  is the case where *no data movement* is done at all during the routing network merger process. Up to 35% of queries could not be answered in such a scenario (even) after the routing networks are merged. This is what will typically happen in previous works like [22] that ignore data synchronization and management issues.

**Overheads:** We also evaluate the overheads of the merger process. From figure 5(a) we notice that in terms of successor changes, we achieve an efficiency  $\eta_s$  varying typically between 0.85-0.9 for various relative sizes of networks studied. This demonstrates scalability of the ring merger process.

Figure 5(b) shows the percentage of data movement in terms of percentage of total data in the system. We notice that if the relative size of the two networks are different, then this leads to more data being transferred in the background data shipment process as compared to merger of two networks of similar size, confirming our analysis of data shipment efficiency. Also, this leads to a slower reestablishment of key to peer bindings and thus slowing down the merger process.

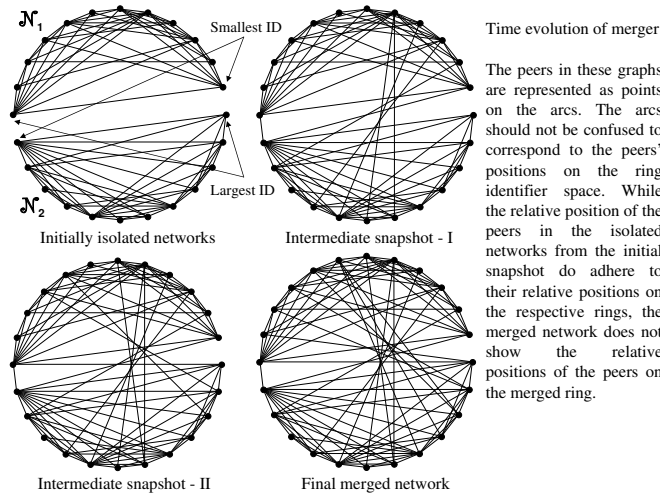
**Merger visualization:** Our simulations were based on discrete time and synchronous interactions among peer pairs. In real life, neither of these will be strictly true, and hence the network rewiring will happen more organically. However the net effect will be similar. In figure 6 we show



(a) Number of successor changes during the merger normalized by the minimum number of changes required for the instantiated rings. The final value in the plots correspond to  $1/\eta_s$ . (b) Data movement for merger of rings with different sizes: Legends are marked \* for peer densities 0.1 and 0.1; ° for densities 0.1 and 0.2 on the identifier space.

**Fig. 5.** Overheads of the merger process for different relative network sizes.

snapshots of the networks' connectivity graph, and see that starting from two disconnected graphs, how once the merger process is started, it evolves into a highly well connected single network.



**Fig. 6.** Evolution of the merger process over time for two very small networks.

## 5 Conclusion

Merging smaller structured overlays seamlessly to organically construct a larger one is essential for large scale deployment of structured overlays. This paper looks into the algorithms for merging the most popular topology - ring structured overlays. The basic ideas are simple and easy to integrate with Chord's self-stabilization algorithms, and the contribution and novelties are as much in defining necessary performance metrics and carrying out experiments to evaluate the proposed

algorithms. In that, we used the information retrieval metric of *recall* to determine the performance of the overlays during and just after the merger process. We also made analytical predictions and validated using experiments the minimal overheads in terms of successor changes as well as data transfer, that will be incurred during the merger process.

This is the first work to deal with merger of ring networks taking into account not only the issues of the routing network, but also data management. The results show that the merger algorithm works reasonably well, but is far from perfect, leaving scope for further improvements; while the theory provides benchmarks for what can be achieved - in terms of overheads.

The temporary poor recall results conform with previous speculations [6] that merger process in a ring will disrupt its functionality or deteriorate its performance in contrast to a tree topology, where mergers can be handled almost transparently ( $\sim 2\%$  failures in the worst cases observed [5]). A more exhaustive comparative study of the two topologies is part of future work.

## References

1. K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. *31st International Conference on Very Large Databases (VLDB)*, 2005.
2. A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM*, 2004.
3. M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *SIGCOMM*, 2006.
4. S. Chaudhuri and V. Narasayya. Index merging. In *ICDE*, 1999.
5. A. Datta. Merging Intra-Planetary Index Structures: Decentralized Bootstrapping of Overlays. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*.
6. A. Datta and K. Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *International Workshop on Self-Organizing Systems (IWSOS 2006)*.
7. P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, 2004.
8. P. Ganesan, P.K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *ICDCS*, 2004.
9. S. Girdzijauskas, A. Datta, and K. Aberer. Structured Overlay For Heterogeneous Environments: Design and Evaluation of Oscar. *ACM Transactions on Autonomous and Adaptive Systems*, To be published.
10. K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the ACM SIGCOMM*, 2003.
11. N. Harvey, M.B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS 2003*, Seattle, WA, March 2003.
12. J. M. Hellerstein. Toward network data independence. *SIGMOD Rec.*, 32(3), 2003.
13. M. Jelasity, A. Montresor, and O. Babaoglu. The bootstrapping service. In *ICDCSW '06: Proceedings of the 26th IEEE International Conference Workshops on Distributed Computing Systems*, 2006.
14. D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.
15. Z. Kis and R. Szabo. Chord-zip: A chord-ring merger algorithm. In *IEEE Communications Letters*, 2008.
16. N. Lester, J. Zobel, and H.E. Williams. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. In *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*.
17. J. Li, J. Stribling, R. Morris, and M.F. Kaashoek. Bandwidth-efficient management of dht routing tables. In *NSDI*, 2005.
18. G.S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS*, 2003.
19. A. Montresor, M. Jelasity, and O. Babaoglu. Chord on Demand. In *IEEE P2P*, 2005.
20. S. Rhea, B. Godfrey, B. Karp, J. Kubiawicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *ACM SIGCOMM*, 2005.
21. 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)*, 2001.
22. T. Shafaat, A. Ghodsi, and S. Haridi. Handling network partitions and mergers in structured overlay networks. In *P2P*, 2007.
23. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, (Technical report version: <http://pdos.csail.mit.edu/chord/papers/>), 2001.