Interconnected Chord-rings

Zoltán Lajos Kis (Corresponding author)

Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics

2 Magyar tudósok körútja, Budapest 1117, Hungary

E-mail: kiszl@tmit.bme.hu

Róbert Szabó

Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics

2 Magyar tudósok körútja, Budapest 1117, Hungary

E-mail: robert.szabo@tmit.bme.hu

Received: November 22, 2009 Accepted: May 21, 2010 DOI: 10.5296/npa.v2i2.271

Abstract

The future of computer networking will be dominated by dynamic, autonomous networks interacting with each other, while constantly forming new topologies by compositions and decompositions. These networks will be governed by distributed management entities, relying on a distributed data storage maintained by the individual members of the autonomous network. Distributed hash tables (DHT) provide a feasible solution for creating and maintaining data storage facilities for such networks. In our previous work we presented Chord-Zip, an algorithm that provides a scalable merger for Chord rings with continuously high data availability, thereby enabling DHTs to be used in dynamic environments of future networks. In this paper we propose a novel architecture for the composition of Chord rings, which maintains the individual ring structures, while providing the same interworking functions as our Chord-Zip merger. This new architecture enables a wider range of network-network interactions, including the movement of nodes between rings, joining or removal of rings from the architecture, and decomposition of a Chord-ring into multiple rings. The proposed architecture can also be used to provide advanced data management features, or replication of data elements based on network topology. It also provides an increased robustness for network failure scenarios, where a merger is not feasible.

Keywords: Composition, Decomposition, DHT, Distributed hash tables, interworking

1. Introduction

Future networks are imagined to be autonomous, where all management decisions are taken based on the common interest of the participating nodes [1]. As the number of mobile and versatile devices is ever increasing, such networks will inevitably become dynamic in nature.

When connectivity is established between networks, they might decide to share some of their resources – such as routing or data storage – with each other in order to foster cooperation. An extreme case of this sharing is merger, where all resources, including management decisions are shared among all participants.

Also these networks can at anytime separate into multiple sub-networks, either due to the lack of connectivity, or because a group of participants decide to lessen the degree of cooperation with the rest of the network.

These network-network interactions can become arbitrarily complex, yet can be described by a few basic interactions. In a composition members of two networks make all their resources available to each other, and subordinate their decisions to the merged group's common interest. In resource sharing, the individual structures of the networks are preserved, and only a subset of their resources can be used by the nodes of the other networks. In decomposition the network is split into two or more networks, where common knowledge might continue to exist in all emergent networks, while sharing of resources becomes limited or is completely nullified among them.

Due to the dynamic nature of autonomous networks, fault tolerant data storage can only be achieved by spreading tasks among all participants. Several overlay networks were designed for providing such a distributed data storage. Unstructured overlays aim at providing a minimal overhead solution at the price of not guaranteeing the correctness of data operations. In contrast, structured overlay networks are designed to provide correctness, while keeping maintenance overhead scalable.

Distributed hash tables (DHT) are a subclass of structured overlay networks, where the neighbourhood relations and the mapping of data to nodes is done based on hash functions. Many implementations of the DHT paradigm coexist, for example [2][3][4], all of which differ only in the routing and storage algorithms used. We base our work on the Chord-ring implementation [2], which is the most accepted by the research community.

Common in the DHT approaches is that they aim at connecting every node into a single, global overlay network. All of these works concentrate on providing scalability and lessening lookup times, providing interactions only between nodes and the global DHT (join and leave), leaving network-network interactions out of scope.

The difficulty of (de)composing hash tables is a consequence of the hashing, what unequivocally defines the overlay structure and data mapping for a given set of nodes. Therefore when DHTs compose, most of the data stored will need to be relocated from one node to another. At the same each node's overlay neighbours need to be completely changed.

We observed that there is nearly enough topological information available at every node in order to execute the merger individually [5][6]. Chord-Zip was designed to provide the missing pieces of information thereby enabling a scalable merger with high data availability.

In this paper we present a novel overlay architecture which enables data sharing autonomous Chord-rings while preserving their original structure. Using this architecture the data management substrate can also reflect resource sharing type of compositions. While this architecture has no additional overhead compared to mergers in stable scenarios, it allows the original Chord-rings to depart at any time, without disturbing the interworking among the rest of the rings.

The rest of the paper is structured as follows. Section 2 gives an overview of the related works on the composition and decomposition of DHT overlays, along with an introduction to our previous work on the Chord-Zip algorithm. Section 3 describes the structure and functionality of the proposed interconnected architecture. Section 4 presents how network dynamism is handled, while Section 5 discusses data management considerations. Finally, Section 6 concludes the work presented in this paper.

2. Related work

Distributed hash tables have been extensively researched recently. The effort is mostly concentrated on the routing and storage algorithms, creating more and more sophisticated global data storage facilities. While these works are of primary importance, they do not provide interactions between mutually coexisting networks. The proposed solution in these situations is to revert to node-network interactions, i.e. disassemble a ring to separate nodes and have each of them interact with the other ring individually.

2.1 DHT Composition

Composition of Distributed hash tables has only been discussed in very few papers yet. These works either try to solve the composition by the reversion to node-network interactions, or by limiting the solution to specific scenarios.

Papers [7][8] investigate the former approach. One of the rings is disassembled into nodes, and these nodes individually join the other ring without coordination. Both works investigate the composition from the aspects of feasibility and algorithmically correctness only, not considering performance metrics, such message complexity, data availability or completion time.

The result of disassembling a ring is that perceived data availability becomes so low, it becomes impossible to rely on the facilities provided during the merger. Also, as a consequence of reversing to node-network interactions, these works also – directly or indirectly – rely on the stabilization mechanism of the Chord-rings, e.g., when joining the other ring. Because the stabilization periods used by these mechanisms are magnitudes higher than the network’s message passing time, the completion time of these algorithms becomes extremely long.

The works of [9][10] are examples for the latter approach. SkipNets are ring structures similar to Chord rings, but with different characteristics. They are aimed at environments where the nodes and their identifiers are hierarchically structured based on their network locations. If the composition only takes place between hierarchical groups of nodes, the merger process is reduced to interactions between two-two nodes at the edges of the merging rings (in terms of address spaces). The latter work provides an abstract algorithmic approach for the merger. While the proposed architecture is correct, no distributed algorithm is provided for building the architecture from existing Chord rings. Therefore this work can only be taken feasible in environments where a central management entity exists on top of the distributed data management.

2.2 DHT Decomposition

In the field of distributed hash tables, decomposition is not a real requirement, as the common interest of nodes dictates the creation of a global overlay network, where all information is available for all participating nodes. Therefore, decomposition is only investigated in the context of autonomous networks, where the cohesion of the network is based on policy and privacy constraints on the individual nodes. In this environment, due to the dynamic nature of node interests, a group of nodes might need to create a new separate network leaving the original one behind.

The idea of decomposing autonomous networks is discussed in [11]. This work yet again uses a fallback solution for the decomposition of DHTs: the nodes of the departing sub-group leave the DHT one-by-one and build up a new DHT by joining in individually. Similar to the works on composition which relies on fallback solutions, this decomposition method will result in high execution time. Also data availability is not investigated, leaving both the original and the new network with low data availability.

2.3 Chord-Zip

We proposed Chord-Zip as an algorithm for merging Chord-rings, free of the limitations of the existing works on composition. Our concerns were providing scalability in completion time, while maintaining high perceived data availability for applications[5].

The merger of Chord-rings must be done in a distributed manner in order to achieve scalability; however participating nodes are not in possession of enough information to individually execute the required changes. The required information is the new references on the merged ring, including the new successor and predecessor, and the list of nodes where data is needed to be transferred from.

The Chord-Zip algorithm is based on our finding, that a node can possess all required information by executing a search request on the other node. Also, once a node has the information, it is able to generate the required information for one another node without executing another search. Exploiting these properties, Chord-Zip works by bootstrapping a single node, and then passing a token around the rings from this node (Fig. 1). Each node upon receiving the token updates its references, initiates necessary data requests, and sends the token onward with updated information.

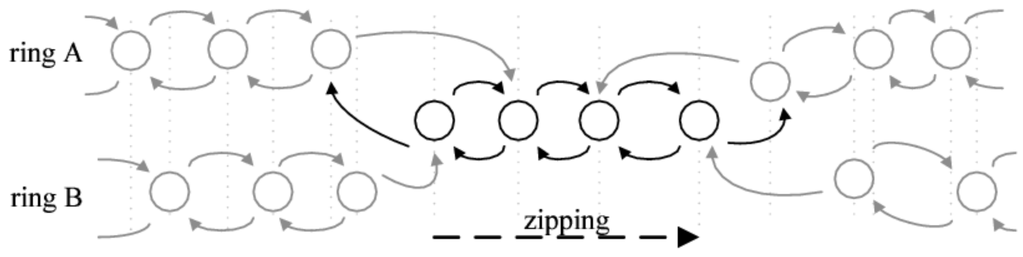

Figure 1. The Chord-Zip algorithm.

The basic Chord-Zip algorithm's completion time is proportional to the total number of nodes (and is independent of the Chord-rings' stabilization period). By bootstrapping multiple nodes in parallel, the total execution time of Chord-Zip can be reduced, inversely proportionally to the number of parallel initiations. We enhanced the parallelization of Chord-Zip by proposing a novel ring-size estimator and a bootstrapping algorithm which allows bootstrapping new nodes during the merger [6]. These features enable Chord-Zip to autonomously control the level of parallelization, in order to complement predefined requirements, while keeping messaging overhead minimal.

3. Interconnected Chord-rings

The interconnected architecture was designed to provide data sharing among Chord-rings. As compared to mergers, where a single Chord-ring is created, the interconnected architecture preserves the structure of the individual rings.

This is not only beneficial when providing data storage for autonomous networks, but can also be used in place of a merger in a number of scenarios. Examples of these scenarios are where the original Chord-rings will need to return to their original structure at a later stage. For example when composing DHTs for a limited time only (e.g. for synchronization), or data sharing among DHTs formed on corporate networks, where the inter-site links are error-prone, and thus each site might frequently need to revert to its local DHT from the global data store.

The interconnected architecture maintains the O(log N) routing property of the Chord-rings, even when extending the routing scope to all participating rings. The maintenance overhead required is the same as in the case of the individual Chord-rings, while node joins and failures require two extra message hops at most; thus providing similar time properties in recovery as the individual Chord-rings.

In the followings we will describe the interconnected architecture with the participation of two Chord-rings for easier understanding. However each of them can naturally be extended to an arbitrary number of interconnected Chord-rings.

3.1 Architecture

The architecture consists of the individual Chord-rings, and additional references between nodes of different rings. The internal architecture of the Chord-rings is preserved: each node maintains a predecessor and some successor references for maintenance and basic routing as well as a set of finger references for achieving scalable routing within its own ring. The interconnecting references are created on top of these original references.

Each node N stores an interconnecting reference to node M in the other ring, if the responsibility intervals of N and M overlap. Figure 2. shows a segment of two rings (straightened for easier drawing), solid lines representing original Chord references (excluding fingers), dashed lines representing interconnecting references. As an example the responsibility interval of node N and M is also shown. These intervals overlap, therefore a symmetric reference between N and M exists.

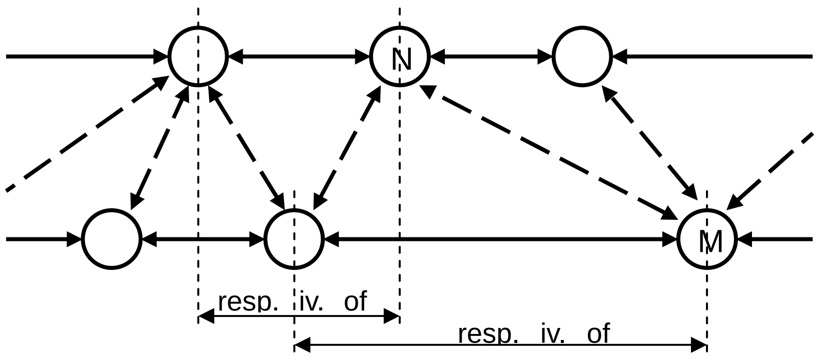

Figure 2. The interconnecting references.

Because of the symmetrical property of overlapping intervals, the references will also be symmetric, i.e., if there is a reference from N to M, there will also be a reference from M to N. Nodes store the overlapping intervals along with the references, so they can directly forward requests to the responsible node of a given key on the other ring.

For every node there will be at least one interconnecting reference, referring to its alternative successor, the node that would be the successor of the given node on the other ring. There is no upper limit on the number of references of a single node. However, as all references are formed between nodes and their alternative successors, the total number of these symmetric references within the architecture will always equal to the number of participating nodes.

3.2 Routing

Routing in the interconnected architecture can have a ring-local or global scope. Local routing is done by the original Chord algorithm, without modifications: a request is forwarded to the responsible node via finger and successor references. Routing in a global scope also starts by routing the request to the responsible ring locally. The responsible node in turn might extend the request to global scope. This is achieved by forwarding the request via the interconnecting references. In case of multiple Chord-rings in the architecture, the request is forwarded on all references in parallel.

As the interconnecting references directly connect a responsible node to its responsible counterpart in the other ring, extending to global scope will only require an extra single hop. Thus, the O(log N) property of the request routing is not modified when using global scope instead of local.

The existence of the interconnected architecture is transparent for applications. They continue to send requests to their local Chord entities. These requests will be forwarded to the responsible node on the ring as a local search. Write requests are executed on the local responsible nodes, and no further action is taken. Read requests are however extended to global scope, when needed. If the requested key can not be found on the local responsible node, the request is forwarded to the responsible node of the other ring via the interconnecting reference. In case of multiple Chord-rings, the read request can be forwarded to all the other rings in parallel.

Multiple, conflicting responses can be handled in multiple ways. There are two trivial solutions: the node can either select one of the values (e.g., the latest one if a timestamp is available), or it can group all responses into a single reply message, leaving conflict resolution to the application. In case of cooperating autonomous networks, where same keys represent the same semantic meaning, the collected values can be aggregated into a single value before sending the response back.

3.3 Maintenance.

In the interconnected architecture only the individual Chord-rings are maintained actively. The maintenance is done by running the original Chord maintenance algorithm within each ring. Message sequences of the algorithm are not modified, but the data sent within these messages is extended. Nodes participating in the interconnected architecture not only pass their successor lists to their predecessors, but also their interconnecting references, including the corresponding overlapping responsibility intervals. References of the successors are also distributed during maintenance, which will result in each node having an up-to-date reference list for all nodes in the successor list.

No active maintenance is used for the interconnecting references, i.e., no keep-alive messages are sent between nodes referencing each other. A reference is created during the interconnect architecture creation (see Section 4.1), or as a result of the internal Chord-ring maintenance algorithms. References are deleted when nodes leave the network, or when a reference is found to be invalid and is not possible to be fixed.

When a node joins a Chord-ring, it is bootstrapped based on the original Chord algorithm. Its predecessor and successor nodes are set, and the node receives its successor list as part of the maintenance algorithm. As the joining node inherits part of the responsibility interval of its successor, only the successor's interconnecting references need to be taken care of. If a referenced node's interval is completely taken over, the newly joined node sends an update message to the referred node, while creating the reference on itself as well. At the same time the successor invalidates its own reference. If the referred interval is separated by the newly joined node, the interval must be separated, where one segment will belong to the original successor and one to the newly joined node. Both nodes send an update to the referred node, the new one creating a new reference, the successor updating the overlapping interval. As both nodes are in possession of the same information, no extra coordination is needed between them, other then the modified maintenance algorithm's messages.

In case of a node failure the failure is detected by the node's predecessor on the Chord-ring as the maintenance algorithm times out. The predecessor will possess the list of the failed node's references as well as its successor (which will be the detecting node's new successor). Thus the predecessor can send a notification to all referenced nodes on the failure indicating the new successor. In turn the notified nodes can update their references, also notifying the referenced nodes.

In Figure 3. greyed lines represent references lost because of the failure of node N. Weak dashed lines show notification messages, strong dashed lines the affected interconnected references (straight ones modified, curved ones created). When node A detects the failure of node N, it sends notifications to nodes C and M, indicating that N failed and its successor was B. As a consequence, C creates a new reference to B (for the interval between A and C) and sends a create request to B. Also M extends its overlapping interval with B (to the interval between C and B), and sends an update on this to B.

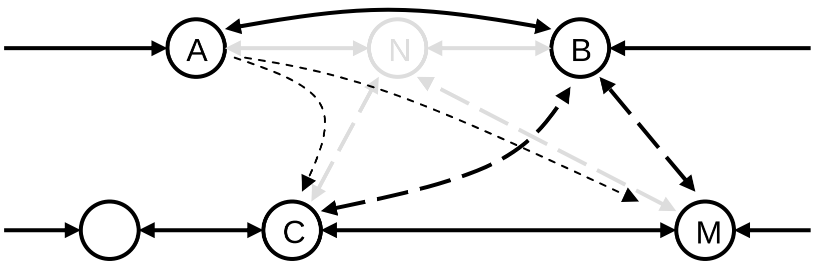

Figure 3. Example of handling node failures

If a reference is found to be invalid by a node, it will try to see if it can contact the successor of the failed node, which is the potential inheritor of the lost reference. This successor will be referenced by one of the successors of the detecting node. In case the successor is referenced, the reference is updated, and the request is forwarded on the updated reference again. Otherwise the invalid reference is dropped, and will later be updated by the maintenance algorithm of the ring of the failed node.

Nodes leaving the Chord-ring are handled in a similar manner to failed nodes, however the notification is sent to the successor by the leaving node itself, this way increasing the responsiveness of the architecture.

These basic maintenance steps enable transition of nodes from one ring to another. By sequentially executing the leave and join operations, a node can switch between rings of the interconnected architecture. As it already has a reference to its alternative predecessor and successor (in the form of interconnecting references), the joining procedure does not involve an initial search: the location of the node on the other ring is already known.

As the maintenance of the interconnect architecture relies on the internal Chord maintenance algorithm, it can only survive failure scenarios a Chord ring itself would. This means that the architecture recovers if the number of mutually failing nodes is not higher than the length of the successor lists propagated by the Chord maintenance algorithm.

Incorporating active maintenance on the interconnect references could increase the robustness of the network, but we prioritized on the maintenance overhead in our work, i.e., our aim was to have no more maintenance work involved in the architecture as in the case of separated (or merged) Chord rings, where the maintenance overhead is proportional to the number of nodes.

4. Network interactions

The interconnected architecture is formed by a set of individual Chord-rings with a similar algorithm to merger. Also Chord-rings can join this architecture at a later point, such as a Chord-ring can be merged to an already merged set of rings. However the architecture supports further network interactions, such as the departure of a Chord-ring. Furthermore this architecture provides a way for a Chord-ring to be decomposed into multiple rings, either as member of an interconnected architecture, or individually (thereby forming an interconnected architecture).

4.1 Creating the interconnect architecture

We have already discussed that in the interconnected architecture there is one reference from each node to its alternative successor. As the Chord-Zip merger algorithm was based on the idea of propagating the alternative successors to each node via tokens, that algorithm can be reused with minor modifications.

The first step in the creation of the architecture is bootstrapping an arbitrary node. This is done by initiating a search for the node’s alternative predecessor and successor on the other ring. First a query is initiated for finding the node responsible for the bootstrapping node’s address, the result of which will be the node’s alternative successor. Then, the alternative successor is queried for its predecessor, which will naturally be the bootstrapping node’s alternative predecessor. Once the bootstrapping is finished, the node will act as if it received a token containing the alternative predecessor and successor, as described below.

A node upon receiving a token, reads the contained alternative predecessor and successor. From these data the overlapping responsibility interval with the alternative successor can be calculated. The lower bound of the interval is either the node's predecessor or its alternative predecessor, depending on which one is closer to the node itself. The upper bound will always be the node in possession of the token. Once calculated, the node registers the reference along with the overlapping interval, and sends a notification to the alternative successor in order to create a back-reference to the node for the same interval.

Finally the node sends the token, and sends it to the next closest node: either its successor, or its alternative successor. In the former case, the contents of the token are untouched, because then the node's successor must have the same alternative references as the node itself. In the latter case, when the token is sent to the other ring, the alternative predecessor is set to the current node, and the alternative successor is set to the node's successor. See Figure 4.

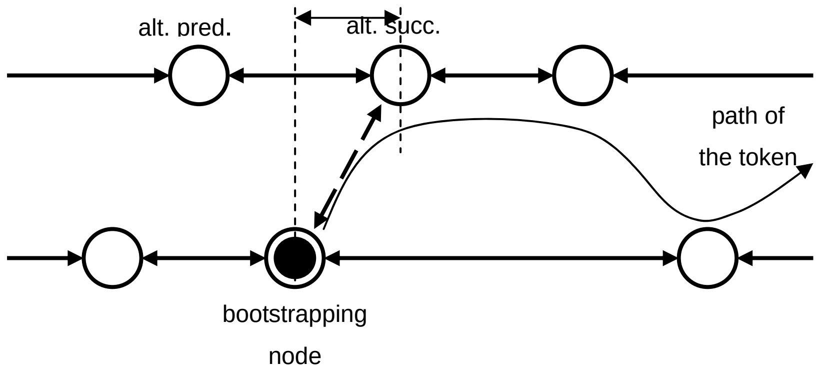

Figure 4. Bootstrapping node

The interconnected architecture is completed when the token reaches back to the bootstrapped node. By this time all symmetric interconnecting references have been created.

The above presented algorithm's completion time is linearly proportional to the total number of nodes. But, as it is algorithmically equivalent to the Chord-Zip algorithm, the parallelisation procedures discussed in [6] can directly be applied to this algorithm as well thus improving scalability.

4.2 Chord-ring joining interconnected rings

When a Chord-ring needs to join an interconnected architecture, missing references need to be created. These references go from nodes of the ring to the nodes in the interconnected rings, and vice versa, in a symmetrical manner.

From an algorithmic perspective, this scenario is identical to the creation of the interconnected architecture, treating the rings as a single Chord-ring. To achieve this, nodes of the interconnected overlay will use their closest references (own or alternative predecessor and successor) instead of using their own ones. This is possible as the interconnecting references will always refer (among other nodes) to the node's alternative predecessor and successor.

During the bootstrap phase, when the search is initiated on an arbitrary node of the interconnected rings, the request must be forwarded via interconnect references to the node that is the alternative successor of the bootstrapping node taking all interconnected rings into consideration. Also, when the alternative successor is requested for its predecessor, it must reply with the closer one of its own or alternative predecessor. During the token passing phase, the nodes of the interconnected rings will have to use the closer of their own or alternative successors as both next hops and alternative successors.

4.3 Leaving or dismantling the interconnected architecture

Each participating Chord-ring is autonomously maintained, so a ring will remain intact if it loses connection to the other rings, i.e., its interconnecting references become invalid. Therefore no action is to be taken either when a ring departs, or when the complete architecture falls apart.

The interconnecting references will automatically be deleted the first time they are used and their invalidity is detected. If the leaving or dismantling is planned by a higher level management, the dissemination of this decision can be done in order to actively delete the references. The deletion can be done individually on the nodes; therefore an arbitrary dissemination algorithm can be used. The details on the decision and dissemination logic are out of scope for the present paper.

4.4 Decomposing a Chord-ring

When describing the maintenance of the interconnected Chord-rings, we already showed that this architecture provides a convenient way for a node to move from one ring to another. This architecture can also be used to enable the decomposition of a Chord-ring into multiple rings. The decomposing ring can already be part of an interconnected architecture, or can be an individual ring, thus using the architecture as an intermediary step during the decomposition.

The details of the management and coordination of the decomposition are out of scope for the present paper. We assume that as the result of these actions a set of nodes participating in the decomposing Chord-ring will be appointed to depart and create a new ring. This set does not need to contain all nodes to depart: once the appointed core set created the new Chord-ring, the rest of the nodes can move over from the decomposing ring to the new one. At this point the original rings and those composed of the core set form a correct interconnected architecture. As a result, moving nodes already know their alternative predecessors and successors (based on the interconnecting references). Therefore moving a node from one ring to another does not involve an initial search operation, resulting in constant node-moving times.

As all nodes of the new Chord-ring are known, their predecessor successor and finger references can be set by the management, thereby creating a proper Chord-ring. At the same time each one of these nodes can leave the original ring, as described in Section 3.3.

Besides, the interconnect creation algorithm is executed in parallel between the newly created and old rings (or in case the ring was part of the interconnected architecture, the algorithm is run between the architecture and the new ring). The algorithm can be started on all departed nodes concurrently. These nodes already have references to their original successors from the old ring; therefore these references can be used as the start of the bootstrapping process. Exploiting this feature, the execution time of the bootstrapping process becomes constant as compared to the original logarithmic time which involves a search.

As a result of the decomposition the remaining ring and the newly created ring will take part in an interconnected architecture. The advantage of incorporating the architecture into the decomposition process is that, due to the data sharing, the applications will not perceive data loss on the newly created ring. Also it allows the possibility of saving the information from the old ring to the new one (as described in Section 5) before finally separating the two rings, by dismantling the interconnected architecture. The rings at this point are already maintained individually; therefore the interconnected architecture can be dismantled without further action.

5. Data management

The proposed data handling for the interconnected architecture is based on minimizing message overhead while providing global scope for the applications. By allowing an increased overhead, the scope of the requests can be changed, and new functions can be introduced in order to provide a higher level of resilience.

5.1 Scope

In the interconnect architecture, pieces of data are stored on the individual rings according to the original Chord algorithm. The architecture is transparent for applications: they use a single Chord API to initiate requests. Write requests are only handled on the local ring. Read requests are handled on a global scale, by forwarding requests to the other ring via the interconnecting references, if a local copy is not found.

A drawback of the transparency is that an application has no means to communicate whether a request – either read or write – has a ring-local or global scope. Therefore the architecture must handle all requests as described above. If the backward compatibility towards applications can be broken, the API towards applications can be changed, enabling them to indicate the desired scope of their requests. Thereby requests could only be extended to a global scope when explicitly requested.

Providing explicit local and global scope on data for applications has a number of potential use cases. Restricting reads to the single rings for some keys enables the reuse of the same keys in separate rings for ring-local data. Enforcing write requests to global scope allows explicit replication of data to all Chord-rings. Given the set of rings are created on separate segments of the physical topology, this also enables topological redundancy.

In overall, the enabling applications to explicitly indicate scope, reduces the bandwidth usage, especially on links connecting separate Chord-rings, and also provides a means to increase robustness by replicating data.

5.2 Replication

The basic operations described so far fulfil the data sharing requirement, but provide no robustness. Robustness can be achieved by also forwarding write requests on the appropriate interconnecting references, thereby duplicating all inserted data to all participating rings. As a result each piece of inserted data will be available in both rings after an incidental network split.

If data availability is required on all rings, the architecture is also capable of actively replicating data from one ring to another when the Chord-rings are interconnected. This is done similar to the data transfers of the Chord-Zip algorithm: when a node initiates the creation of a symmetrical reference for a responsibility interval, the nodes also send each other a copy of their key-value pairs falling into the given overlapping interval. Conflicting keys can be handled differently, either on a network or on an application level.

The replication can also be initiated at any time during the lifetime of the interconnect architect. The replication can be handled individually by nodes (and their referenced nodes), therefore it can be requested on all nodes by using arbitrary procedures. The nodes will send each other their key-value pairs over the symmetric references. This way, in case of a planned separation of networks, data security can be ensured.

5.3 Robustness

The interconnected Chord-rings based replication can potentially replace the replication facilities provided by various Chord-ring implementations. In most Chord-ring based replications, data elements are replicated to successors. Therefore in case of a node failure the data is already available on the new responsible, but need to be replicated to one another successor to maintain the robustness. In the interconnected architecture data is stored at responsible nodes of participating rings. After a node failure the new responsible can reacquire the lost data from any of the other rings. Figure 5 shows an example of these robustness facilities. The left figure shows a single merged ring, where a given key is replicated to the two successors. The right figure shows an interconnected architecture, where the key is stored in each separate ring (interconnecting references are omitted).

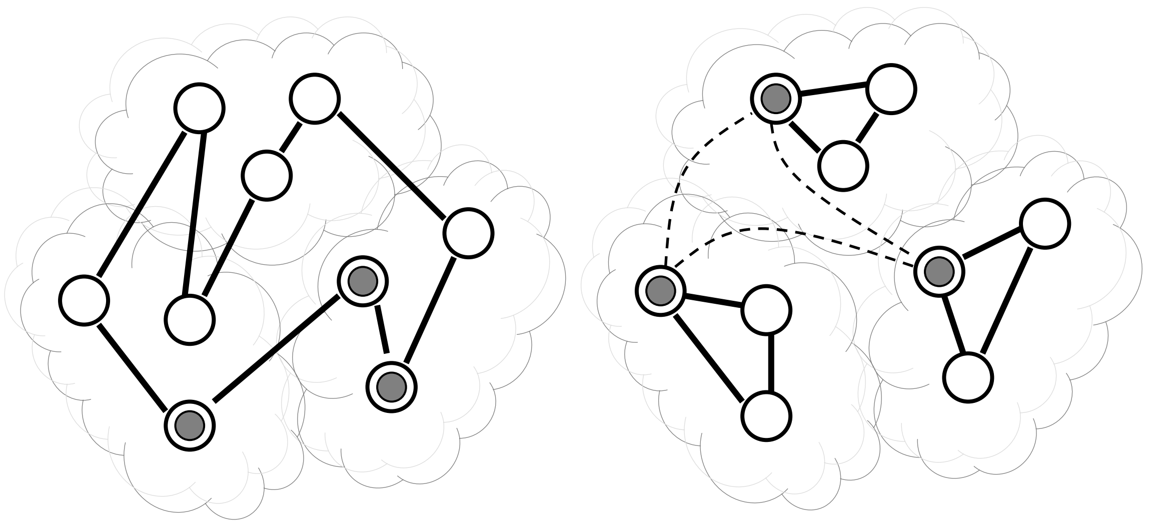

Figure 5. Redundancy provided by Chord implementations
and the interconnected architecture

Furthermore, in a Chord-ring implemented replication scheme, one can only ensure the number of replicas available in the network, but cannot influence the physical location of these replicas. Therefore this solution cannot guarantee data availability after network splits. When using the interconnected architecture for robustness, a replica will be available in each ring. The network management, by locating the Chord-rings along potential failure boundaries, can ensure that all data will be available in each separated ring in case of a network split (see Figure 5.).

6. Conclusion

In this paper we proposed a novel architecture, the interconnected Chord-rings, which enables data sharing among Chord-rings, while preserving the composing ring's autonomy. This is achieved by virtually merging cord rings into an interconnect overlay in which read and search operations are performed, while limiting write requests to the local rings. The composition, decomposition, maintenance and data management algorithms and the robustness property of the proposed architecture were discussed in details. The architecture enables Chord-rings to also be used in the context of autonomous networks, where a lower level of cooperation needs to be supported by the data management substrates. The architecture can also be used to provide DHT-related features, such as decomposition, or topological redundancy.

We believe that the presented work can directly be reused and further investigated in the field of autonomous networks as well as in the context of distributed hash tables.

References

[1] Kersch, P., Szabó, R., Kis, Z. L., Erdei, M., Kovács, B., “Self Organizing Ambient

Control Space – An Ambient Network Architecture for Dynamic Network Interconnection", 1st ACM workshop on Dynamic interconnection of networks, Cologne, Germany, September 2005.

[2] Stoica, I., Morris, R., Karger, D., Kaashoek, F. M., Balakrishnan, H., "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", SIGCOMM, San Diego, USA, August 2001.

[3] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., "A Scalable Content-Addressable Network", ACM SIGCOMM San Diego, USA, 2001.

[4] Maymounkov, P., Mazieres, D., "Kademlia: A Peer-to-peer Information System Based on the XOR Metric", IPTPS '02, Cambridge, USA, 2002.

[5] Kis, Z. L., Szabó, R., "Chord-Zip: A Chord-ring Merger Algorithm", IEEE Communications Letters, Vol. 12. Issue 8., 2008.

[6] Kis, Z. L., Szabó, R., "Scalable Merger of Chord-rings International Journal of Communication Networks and Distributed Systems", Vol. 4., Issue 4., 2010.

[7] Datta, A., Aberer, K., "The challenges of merging two similar structured overlays: A tale of two networks", IWSOS workshop, Passau, Germany, 2006.

[8] Heer, T., Gotz, S., Rieche, S., Wehrle, K., "Adapting distributed hash tables for mobile ad hoc networks", PerCom workshop, Pisa, Italy, 2006.

[9] Harvey, N., Jones, M., Theimer, M., Wolman, A., "Efficient Recovery From Organizational Disconnects in SkipNet", IPTPS '03 workshop, Berkeley, USA, 2003.

[10] Ganesan, P., Gummadi, K., Garcia-Molina, H., "DHTs with hierarchical structure", Distributed Computing Systems, Phoenix, USA, 2004.

[11] Cheng, L., Ocampo, R., Jean, K., Galis, A., Simon, Cs., Szabo, R., Kersch, P., Giaffreda, R., "Towards Distributed Hash Tables (De)Composition in Ambient Networks", Large Scale Management of Distributed Systems, Book, October 2006.