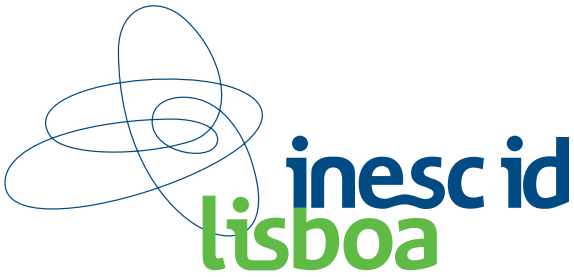

Technical Report RT/12/2013

Rollerchain: a DHT for Efficient Replication

João Paiva
INESC-ID/IST
jgpaiva@gsd.inesc-id.pt

João Leitão
INESC-ID/IST
jleitao@gsd.inesc-id.pt

Luis Rodrigues
INESC-ID/IST
ler@gsd.inesc-id.pt

June 2013

Abstract

In this paper we present Rollerchain, a novel Distributed Hash Table that offers efficient data storage through the combination of gossip-based and structured overlay networks. The unstructured component maintains clusters of fully connected nodes, where each cluster acts as a virtual node in the structured component. This architecture simplifies the management of data replication and balances the load among nodes in the system. We have implemented a prototype of Rollerchain that we have used to experimentally validate its performance against other state of the art solutions.

Rollerchain: a DHT for Efficient Replication

João Paiva, João Leitão and Luís Rodrigues
joao.paiva@ist.utl.pt, jc.leitao@fct.unl.pt, ler@ist.utl.pt

INESC-ID, Instituto Superior Técnico, Universidade Técnica de Lisboa

Abstract. In this paper we present Rollerchain, a novel Distributed Hash Table that offers efficient data storage through the combination of gossip-based and structured overlay networks. The unstructured component maintains clusters of fully connected nodes, where each cluster acts as a virtual node in the structured component. This architecture simplifies the management of data replication and balances the load among nodes in the system. We have implemented a prototype of Rollerchain that we have used to experimentally validate its performance against other state of the art solutions.

1 Introduction

Distributed Hash Tables (DHTs) [1,2,3,4] are *structured overlays*, i.e., overlays where nodes organize themselves into a predefined topology that supports routing. As a result, DHTs can efficiently map a key to a peer active in the system (named the key owner). This functionality allows to implement store/lookup operations of key/value data pairs in a distributed and scalable manner. On top of this basic functionality it is possible to build several types of large-scale distributed applications and services, such as resource-location [5], publish subscribe [6], multicast [7], distributed storage [8], among others [9,10]. As in any other system, in DHTs nodes can voluntarily join, leave, or simply fail. Furthermore, in open large-scale systems, these membership changes can happen at a fast pace, a phenomenon known as *churn* [11]. To avoid key/value pairs from being lost when a node leaves, they need to be replicated in the DHT. Therefore, key replication is a fundamental aspect of the DHT operation.

The cost of replication has been closely analysed in [12], which shows that the enforcement of data replication degrees accounts for a significant part of the bandwidth consumption in a distributed storage system and effectively limits the amount of data that can be stored in a DHT. Therefore, the design of DHTs topologies and replication strategies that mitigate these costs is a matter of extreme practical relevance. This paper addresses exactly this problem. As we show in the evaluation section, our architecture can achieve significant savings when compared to previous solutions from the literature. Furthermore, these gains are not achieved by sacrificing or trading off fault-tolerance. On the contrary, experimental results show that our architecture, maintains the keys reachable even under heavy churn while previous approaches fail to preserves data availability in such conditions.

The problem of replicating key-data pairs in DHTs has been tackled using three main strategies. One consists in keeping copies of the data in r neighbours (typically the r successors in the DHT ring) of the key owner [1,2,10]. Other consists of adding salt to the key to deterministically store the value in r other nodes on the DHT [9,13,14,15]. In Path Replication [7], replicas are created as a result of query processing, by caching the results in the nodes that forwarded the query. A cost comparison of some of these approaches was presented in [16]. Later in the text we provide arguments that highlight the limitations of these approaches (the bandwidth consumption that results from these limitations is quantified in our experimental work). Finally, there is a recently proposed fourth alternative that integrates strongly consistent consensus-based replication with DHTs. Scatter [17] is a key-value storage with support for data replication, that relies on Paxos [18] for maintaining a DHT ring composed of strongly consistent replication groups.

In this paper we present an alternative implementation strategy to build a topology similar to that of Scatter [17]. We have named our implementation Rollerchain. Contrary to Scatter, that heavily relies on consensus to implement every operation, and therefore has higher signalling costs and may block under heavy churn, Rollerchain heavily relies on gossip-based mechanisms. This alternative is interesting, because gossip-based approaches have proven to be a reliable strategy to cope with scenarios where pure structured approaches fail [19]. While Rollerchain provides weaker guarantees, by using a robust gossip scheme and a clever topology maintenance, we show in the evaluation that it is able to avoid the loss of keys even under heavy churn despite its best-effort model.

The rest of this paper is organized as follows. Section 2 discusses related work in more detail. Section 3 provides a brief overview on the building blocks of our solution, Section 4 describes the operation of Rollerchain, and Section 5 presents the results obtained through extensive experimental evaluation. Finally, Section 6 concludes the paper.

2 Related Work

2.1 Data Replication on DHTs

Neighbour Replication (also known as leaf set replication) keeps copies of each key in the r neighbours of the key owner. A significant advantage of Neighbour Replication is that when its neighbours change, the key owner may trigger the creation of new replicas. Since each node keeps a different set of replicas, bookkeeping becomes costly if one attempts to use a flexible scheme where the number of replicas fluctuates. Therefore, most schemes use some variant of eager replication, where replicas have to be created when nodes fail and moved when nodes join. This becomes very expensive in terms of bandwidth as demonstrated in [12]. Furthermore, depending on the routing scheme used in the DHT, Neighbour Replication may not perform a fair load distribution, as some replicas are more likely to be hit by queries. Multi-Publication stores r replicas of each data item in different positions of the DHT. Then, some mutual monitoring scheme

needs to be implemented to detect the departure/failure of a node containing a replica and subsequently restore the replication degree. The main advantage of Multi-Publication is that it offers very good load balancing properties, as multiple queries may be diverted to different regions of the DHT. On the other hand, monitoring becomes extremely expensive, because it needs to use DHT routing and a node may be forced to monitor a different set of nodes for each object that it stores. Other interesting data replication strategies for DHTs can be found in [20] and [21]. Although also relying on cluster-based techniques, these works rely on algorithms that impose additional restrictions to topologies, making them less flexible and limiting their scalability.

Finally, a number of recently proposed overlays take a different approach to replication [17,22]. These overlays create self-contained replication groups of nodes which act as single nodes in the DHT. Routing is performed at the group level and not at the physical node level, allowing the system to fine-tune the replication sets. So, even though they are based on consistent hashing (and hence metadata-less approaches), these overlays allow to decouple the management of replication from management of the overlay topology. In Group-based DHTs, each node is a logical entity, materialized by a replica group of variable size. Contrary to classical DHTs where each node has a pre-determined location in the network (depending on its identifier), in these approaches nodes can join any existing replica group. All members of each group coordinate to act as a single node in a higher layer, defining a DHT. Since replication is decoupled from the DHT layer, the replication degree of individual groups can vary without affecting the DHT's structure. As a result, consistent hashing is used to place data items into groups, and data is then replicated among all nodes that belong to that group. Our paper provides an alternative implementation of this approach.

2.2 Load Balancing on DHTs

Our work leverages on the data replication to improve load balancing. One fundamental problem in DHTs is that node's identifiers or, most likely, keys, may not be uniformly distributed in the address space. As a result, some nodes will be required to maintain (and answer queries for) many items while others may be relatively offloaded. A common technique to circumvent this problem is to use virtual servers [8,23]. In this technique, each physical node joins the DHT using multiple identities; each identity represents a virtual node maintained by that physical node. On the other hand, having multiple virtual identities requires each node to maintain more routing information and monitoring more overlay neighbours, which may impose an excessive overhead. Also, this strategy amplifies the effect of churn, as the departure of a single physical node causes the simultaneous failure of multiple virtual servers. Other approaches rely on making a guided choice of node identifiers at join time, to select positions in the Identifier ring such that the load is evenly distributed among all nodes. In order to achieve this, works such as [24] and [25] use probes in the system to determine the best identifier to use. These schemes allow to balance the load of

object storage among all nodes in the system without requiring additional routing information, by increasing the cost of join operations. However, they may create a non-uniform distribution of nodes in the identifier space, which hinders the performance of some routing algorithms.

2.3 Combining Structured and Unstructured Overlays

As previously mentioned, our solution relies on the combination of structured and gossip-based overlay mechanisms. Some previous systems have already explored this idea, although with different goals and, as a result, with solutions that are structurally quite different from Rollerchain. The work by Ghodsi *et al.* [19] makes the case for combining gossip-based and structured networks, and discusses several examples of successful synergies between both designs. This work claims that through this symbiosis, the current state of the art on DHTs can be improved with new overlay designs that offer better reliability, lower bandwidth costs, or better geometry. Inspired by the work of [26], in [27], Maniymaran *et al.* present an approach that follows this design principle and combines two overlays, a DHT and a interest-based unstructured overlay, at a cost similar to that of building a single one. Kelips [4] is a DHT structured using virtual nodes composed of several physical peers. Unlike Rollerchain, each of these nodes know at least one contact in every other virtual node, creating an one-hop DHT. Kelips supports efficient lookups, but it does not present any solution for data replication and each node stores a pointer to every object owned by its virtual node, a property that severely limits its scalability. Concurrently to our work [28], [29] proposes an architecture inspired on similar principles; however Rollercain uses different techniques to perform joins, merges, and to maintain the routing table.

3 Rollerchain Overview and Building Blocks

Rollerchain dynamically manages the replication groups by combining features of unstructured and structured overlay networks in an integrated design. More specifically, Rollerchain builds a DHT where each (virtual) node is materialized by a small group of peers, the size of which depends on the replication degree, R. These groups are neither static nor defined a priori. Instead, they are dynamically created and maintained by the unstructured component. Acting collaboratively, peers on each virtual node share among themselves the information required to maintain the DHT topology and the data stored by their virtual node. The unstructured component of Rollerchain is responsible for creating and maintaining the virtual nodes. Some of its mechanisms are inspired by Overnesia, an unstructured overlay that aggregates peers in clusters [30]. The structured component of Rollerchain runs the DHT maintenance algorithms. For self-containment, before describing the operation of Rollerchain in detail, we provide a brief overview of Overnesia and of the Chord DHT [1], whose architectures have inspired our design.

Overnesia Overnesia [30] is an unstructured overlay network where nodes self-organize into fully connected clusters which, in turn, are highly and randomly connected among themselves. The target size of these clusters can be configured by the application to a given *targetClusterSize* value. However, the cost of ensuring that every generated cluster produced by the protocol has exactly the same size in highly dynamic and open environments can be prohibitively high. To overcome this challenge, Overnesia instead ensures that the size of clusters is distributed between *minClusterSize* and *maxClusterSize*, with a predominance of clusters with the *targetClusterSize* (evidently, $minClusterSize < targetClusterSize < maxClusterSize$). Each Overnesia cluster is assigned a random identifier that is known by all elements of that cluster. A gossip-based anti-entropy mechanism, in which elements of each cluster periodically and randomly exchange messages among themselves, is employed to ensure that cluster members converge on a consistent view of the cluster membership, despite concurrent joins and failures in the system. Furthermore, this process is used to provide a minimal amount of coordination required to increase the diversity of external links maintained by different cluster members. Note that Overnesia does not offer any DHT support. As a result, the way links are established among clusters does not take any sort of routing requirements into account, other than attempting to maximize the connectivity of the network.

Chord Chord [1] is a widely known DHT that organizes nodes in a ring-like topology. It places objects in specific nodes of the ring using consistent hashing. Chord's ring is created by sorting nodes by their identifier modulus the size of the identifier space. Its maintenance is mostly proactive, such that each node keeps a predecessor and a successor node through periodic maintenance routines. More specifically, each node N periodically queries its successor S in order to obtain the predecessor P of S (naturally, in a stable scenario, we should have $P = N$); should $P \neq N$ be a node with an identifier in the interval $]N\_id; S\_id[$, N will then switch its successor pointer to node P. After this update, N informs P that it is now P's predecessor. This simple routine allows the ring to converge and remain connected even in face of concurrent entry and departure of nodes. Even though Chord's ring would suffice for any node to reach any other node in the overlay, routing messages exclusively through this ring would be very inefficient. Thus, Chord's protocol includes an efficient routing mechanism: each Chord node maintains a *Finger Table*, from which it selects the closest node on the ring to route messages towards their destination. As the Finger Table contains pointers to nodes which are at exponentially increasing distances from the node's position in the ring, this mechanism allows Chord to route messages in $log(N)$ network hops, where N is the total number of nodes (since the overlay distance to the destination can be halved with each hop).

4 Rollerchain Operation

4.1 Definitions and Basic Operation

We opted to preserve the nomenclature of the original Chord paper, where each peer is denoted a *node*. Therefore, in Rollerchain, each peer is called a physical node, or *pnode* for short. The unstructured component of Rollerchain aggregates pnodes in clusters. All pnodes that belong to the same cluster cooperate to behave collectively as a logical virtual node, or *vnode*. A *vnode* is a fully connected cluster of pnodes with a fluctuating size around R (the replication factor) that act as a single node in the structured layer. To facilitate the coordination among pnodes in the same vnode, an anti-entropy gossip-based protocol is executed among them. This protocol allows pnodes to exchange information required for the operation of Rollerchain (which we will incrementally describe), including the membership of the vnode and key/value pairs maintained by its members. Also, to simplify coordination among vnode members in several Rollerchain procedures, the member with the lowest process identifier in each vnode acts as the *vnode leader*. Occasionally, it may happen that more than a single pnode sees itself as the vnode leader. This does not affect the correctness of Rollerchain, as the leader is only used to reduce the signalling costs of the algorithms. Similarly, if no pnode sees itself as the leader, this only delays the progress of the algorithms until the anti-entropy procedure enables one of the members to see itself as the leader. Virtual nodes establish *virtual links* among themselves to create a logical ring. A virtual link between vnode A and B is materialized by establishing links among pairs of pnodes (a_i, b_j) where $a_i \in A \wedge b_j \in B$. The algorithm for establishing and maintaining virtual links is described later in the text. Fig.1 provides a visual representation of Rollerchain's architecture.

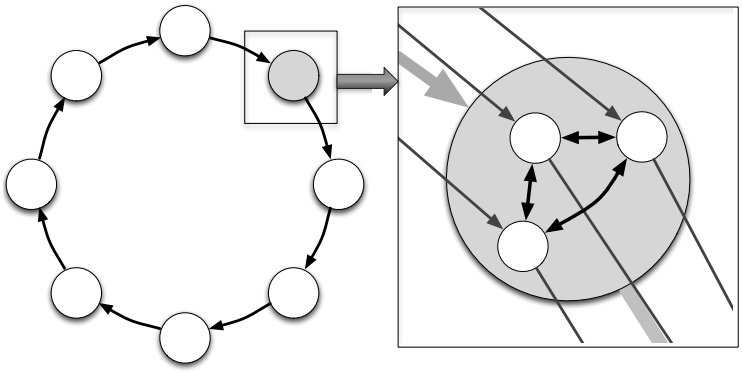

Fig. 1. Rollerchain's Architecture: At the DHT level vnodes form a ring. Each vnode is composed of several pnodes.

4.2 The Unstructured Layer

The unstructured layer of Rollerchain is an overlay composed of virtual nodes, where each vnode stores key/value pairs replicated by all of its members. This replication not only increases the resilience of data, but also allows pnodes to share the load of answering queries for objects stored in the associated vnode. When joining the unstructured layer, a pnode starts a random walk in the network, which probes suitable clusters to join. As one of Rollerchain's goals is to balance the load among the virtual nodes that compose the system, the new pnode chooses the most heavily loaded virtual node found in the random walk to join the network. We define the load of a vnode as the number of key/value pairs it stores, normalized to the number of pnodes it is composed of. This mechanism causes heavily loaded virtual nodes to attract new members in order to share their load.

When a vnode leader detects that its vnode has become too large, it starts a division procedure to halve the vnode into two others with the same size. This division serves not only to reduce the costs of replicating data among its members, but also to reduce the number of objects each member stores. When dividing, the position of the vnode in the identifier ring is critical for the good performance of the system. If the joining vnode just selected a new random identifier as it happens in Overnesia, it would discard all of its data, rejoin the DHT and receive new data from its new successor. Thus, to improve the efficiency of the structured layer, when a virtual node is divided in Rollerchain, one of the newly generated vnodes keeps the identifier of the original vnode (to avoid inducing artificial churn in the structured overlay) and the other generates an identifier that allows it to become the owner of half of the original vnode's key/value pairs.

A vnode merge occurs when the vnode leader detects that its vnode's size is dangerously below the target replication degree. The merge prevents objects from being lost by integrating the vnode with its successor. The vnode leader obtains its successor vnode's membership and broadcasts it to its neighbours, which then change their vnode identifier.

4.3 The Structured Layer

The structured layer of Rollerchain is a double-linked ring composed of virtual nodes provided by the unstructured layer. As described earlier, the vnode identifiers are selected to promote the load balancing of the number of objects stored by each vnode. Therefore, vnode identifiers are not uniformly distributed in the identifier space. In typical DHTs (such as Chord [1], Pastry [2] or Kademlia [3]), node identifiers are assumed to be uniformly distributed in the identifier space. Thus, typical DHT routing mechanisms cannot be applied to Rollerchain. To ensure efficient routing in Rollerchain's identifier space under these conditions, each vnode in Rollerchain has a virtual link to its immediate successors and one finger table with $log(N)$ rows, containing virtual links to distant vnodes in the ring. As it happens in DHTs such as Chord [1] or Viceroy [31], each row in

the routing table represents an exponentially larger jump than the previous row. However, whereas in these solutions the larger jumps refer to the *identifier space*, in Rollerchain each row in the routing table represents an exponentially larger jump in the *number of virtual nodes on the ring*. When vnodes are distributed uniformly in the ring, both schemes generate similar routing tables. However, if some areas of the identifier space have more vnodes, these will appear more frequently in Rollerchain's routing tables, whereas in typical DHTs they would have the same probability regardless of the density of those regions. It is important to notice that even though this routing scheme ensures that our system can route messages logarithmically with the number of vnodes, other routing mechanisms, such as those based on Skipnets [32] or the work presented in [33], could also be employed.

4.4 Layer Interoperation

Virtual Link Creation As described before, virtual nodes cooperate to create a Ring-based DHT. To preserve the logical ring, and to support efficient routing, each virtual node maintains a virtual link to other virtual nodes in the ring. More precisely, a virtual link needs to be maintained to each different virtual node in the finger table (including the successor). Virtual links between two vnodes, say V_A and V_B, need to be materialized by links between individual members of these vnodes (Fig. 2a). For fault-tolerance and load balancing, these links should be distributed evenly among these members. For instance, consider a vnode V_A that has n members and a vnode V_B also with n members. It would be highly undesirable to have a singe pnode a_k from V_A to have n links to each member of V_B (Fig. 2b). It would be equally undesirable if all nodes in V_A establish a link to the same pnode b_k in V_B (Fig. 2c). The ideal solution would be to have each pnode $a_i \in V_A$ to have a link to a different pnode $b_i \in V_B$ (Fig. 2d). In this way, each pnode would be required to maintain a single link and the virtual link would be materialized by n independent physical links. Additionally, this ensures that $n-1$ physical links between V_A and V_B would still be alive after a single pnode failure/departure.

In order to quickly achieve an even distribution of links among members of a vnode, the vnode leader (the pnode with the smallest identifier), say a_0, coordinates a procedure of virtual link creation. This procedure is used when a vnode leader updates the virtual links of its vnode, particularly during vnode division and vnode finger update. We assume that when the virtual link creation procedure is invoked, a_0 is aware of at least one member $b_k \in V_B$. The leader a_0 starts by requesting b_k to return the current snapshot of V_B's membership; we recall that there is an anti-entropy procedure running inside each vnode that keeps such information up-to-date. Knowing the full membership of V_A and V_B, the leader a_0 assigns links among V_A and V_B in a round-robin manner, until all pnodes of V_A and V_B are connected to some pnode in the other vnode (note that Fig. 2 is an over simplification of reality, as V_A and V_B generally do not have the exact same number of pnodes). Finally, this mapping is propagated to

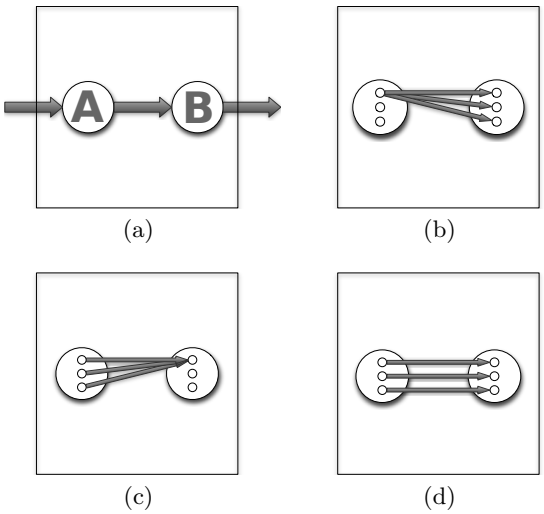

Fig. 2. Management of virtual links among vnodes.

all members of V_A. Each individual pnode a_k then initiates the establishment of link(s) to its corresponding pnode(s) in V_B.

Virtual Node Maintenance The membership of a vnode may change as pnodes join and leave the overlay. As it is crucial for the connectivity of Rollerchain that its ring remains connected despite membership dynamics, such changes require adjustments to the link assignments that materialize the virtual links maintained by each vnode to its successor and predecessor nodes. [1] Consider first the result of a join operation on virtual node V_A by pnode a_k. The pnode may help in materializing the virtual links maintained by the virtual node V_A. To this purpose, and for each virtual link, pnode a_k will attempt to "alleviate" the load of some other members of V_A, by serving as endpoint of some of their links. This can be achieved as follows. The information exchanged in the anti-entropy protocol includes the number of forward and backward links maintained by each pnode for a given virtual link. Therefore, a_k can select a node that has more forward or backward links than the majority of the remaining pnodes and connects to one of their endpoints. When the other pnode finds out about a_k's links, it closes the redundant link. On the other hand, if a_k finds that the number of links is perfectly balanced, the new node just creates a redundant link at random. Consider now the case where a pnode leaves a vnode. The physical links established by that pnode will break. This means that pnodes on the other endpoint of those links will perform the link re-distribution described in the previous paragraph. Finally, consider the case where multiple join and leaves occur in succession. If joins and leaves are perfectly interleaved, a (physical) link is

[1] Note that there is no need to adjust the remaining virtual links in the finger table as those are non-essential to maintain overlay connectivity.

lost in each leave but a new link is created in each join, and the number of links that materializes the virtual link between two vnodes remains constant. However, if bursts of leaves or joins occur, the balance may no longer be preserved. When, as a result of the anti-entropy procedure, the vnode leader detects heavy imbalance in the link distribution for some virtual link, it triggers a *re-balance* procedure. This rebalance procedure consists of sending a balanced mapping of physical links for the virtual link to the pnodes causing the imbalance, so that they may adjust their links accordingly.

Virtual Node Division Vnode division is performed in such way that the two new resulting vnodes have similar size and load. For this purpose, one of the vnodes preserves the identifier of the original vnode and the other vnode becomes its predecessor, by assuming an identifier that causes it to become the owner of half of the key/value pairs of the original vnode. This allows Rollerchain to dynamically adapt to the distribution of keys in the identifier space. Virtual node division is controlled by the leader of the original vnode, that sends a message to all members with the identifiers of the new vnode, the membership of each vnode, and a new assignment for the virtual links maintained between them. This division procedure ensures that the new vnodes are neighbours in the DHT, what contributes to savings in the number of keys moved during this operation, as nodes will only have to locally discard object/key pairs.

Virtual Node Merge In a steady-state scenario, vnodes are stable, as new pnodes tend to replace failed/departed pnodes. However, as a result of multiple leaves and failures, the size of a vnode may become below the desired replication level for the key/value pairs. This happens when a vnode has very few keys in proportion to its number of members while other vnodes remain heavily loaded in the system. Since the join procedure looks for vnodes that are highly loaded, in order to offer a better load balancing, a vnode with few keys (and therefore, not heavily loaded) may be disregarded by joining pnodes. When a vnode size becomes too small, it is merged with its successor. This ensures that all nodes that are part of the two merging vnodes retain their key/value pairs. The anti-entropy in the resulting vnode will ensure that those keys will be later replicated by all remaining members. Similarly to other mechanisms, the merge is started by the leader of the merging vnode. In highly dynamic environments, different nodes may see themselves as the leaders of the same vnode, and send out conflicting orders regarding vnode division or merge. However, the system is designed to tolerate these scenarios, as a merge order always supersedes a division order. Thus, even though such orders may create temporary inconsistencies (which are unavoidable in large asynchronous systems), the topology will eventually converge to a correct configuration.

4.5 Key/Value Pairs Replication

The replication of key/value pairs among the members of a vnode is performed using a combination of eager and lazy replication schemes. When a key/value

pair is inserted in a vnode, eager replication is used. The pnode that receives the request uses a best effort application-level multicast primitive to replicate the pair among the other members of its vnode. Subsequently, replicas are maintained using a lazy replication scheme that leverages on the anti-entropy protocols executed among vnode members. Processes include a hash of the keys they own for their current interval of responsibility in the anti-entropy messages. If (as a result of an anti-entropy exchange) a pnode discovers that its hash differs from that of a neighbour, they exchange their full list of keys, so that both processes may request their missing key/value pairs from each other. To maintain the correct keys at each vnode, each pnode periodically checks the keys it stores. When a pnode detects it is storing objects for which the keys should be owned by its successor (the keys are between its vnode identifier and its successor vnode identifier) or its predecessor (the keys are not between its identifier and its predecessor identifier), it transfers the content to the correct vnode and deletes it locally. The data is sent to a pnode in the other vnode, which in turn sends the hash of its keys to all the remaining elements of its vnode, so that they may request their missing key/value pairs (if any). As it happens with other replication mechanisms for which there is no master copy of the data (e.g. [34]), this mechanism does not provide strong consistency among the replicas. The anti-entropy mechanism running in each vnode guarantees that eventually all nodes will locally store all the key/value pairs.

4.6 DHT routing

DHT routing in Rollerchain is similar to Chord routing with some twists. Logically, lookups follow virtual nodes, using the virtual links maintained in the vnode's finger table. In reality, lookups are implemented by individual pnodes. When the lookup arrives at a pnode, it uses its own finger table to select a pnode to be the next hop for the lookup. If the pnode cannot contact the next hop, the lookup is re-routed to another pnode in its own vnode, which attempts to forward the lookup using one of its own links. This is similar to the re-route mechanisms used by DHTs that have routing tables with alternate paths (for instance, Pastry [2]). Due to the redundancy in the virtual link maintenance, it becomes very hard to undermine routing in Rollerchain. As it happens in Chord, all lookups end in the predecessor of the key, which return their successor as the owner of the key. In Rollerchain, when lookups reach one pnode in the predecessor vnode of the target key, it returns a random successor pnode (we recall that each pnode knows the composition of its successor vnode through the anti-entropy protocol). This allows the query load to be equally distributed by all nodes despite some (possibly temporary) imbalance in incoming links.

5 Evaluation

In this section we present experimental results that validate the design of Rollerchain. All results reported in this paper were obtained using the Peersim simulator [35] event-based engine. The system was populated with 50.000 objects and

is composed of 10.000 nodes. To extract comparative measures, we used Chord with the Neighbour Replication scheme and Chord with Multi-Publication (as described earlier). We evaluate the performance of Rollerchain in terms of bandwidth efficiency and load balancing.

5.1 Experimental Parameters

All Chord configurations used a replication factor of 7 (in Neighbour Replication, each key is replicated in the owner and in 6 of its successors). Rollerchain was configured with a *minClusterSize* of 3 and a *maxClusterSize* 8, which we experimentally determined results in vnodes being composed, on average, by 7 pnodes. This means the average replication factor of all replication techniques is the same. Furthermore, Rollerchain's routing table size was set to 11, to ensure that both protocols have a similar number of (distinct) fingers, creating routes with the same number of hops (15 on average). The anti-entropy mechanism of Rollerchain was also executed as often as the replication maintenance routines of the other replication schemes. The results presented are the average of, at least, 5 individual simulations for each scenario.

5.2 Communication Overhead under Churn

In order to demonstrate the advantages of Rollerchain in terms of objects transferred between nodes and the impact of combining two overlays, this section presents experimental results for the communication overhead of each system in a (highly) dynamic scenario. To this end, we have conducted experiments as follows. All overlays were initialized by having nodes join the system one at a time. After a stabilization period, churn was induced for 10.000 consecutive simulation steps (one step corresponds to up to 5 Round Trip Times). In each step, c nodes were concurrently removed and c new nodes were added (c is dubbed *churn rate*). We performed experiments with churn rates of 1, 10, and 100. By using increasing churn rates we can assess how each overlay and each replication scheme responds to increasing dynamics in system membership.

Objects transferred between nodes

One of the main goals of Rollerchain is to take advantage of its flexible replication degree to reduce the bandwidth required to maintain objects available when compared with previous solutions. Table 1 shows the average number of objects transferred per node during the course of the simulation and the percentage of objects reachable at the end of the simulations for the same churn rate.

We first direct the reader's attention to the $c = 1$ scenario, where all topologies are able to preserve the reachability of all keys. In the remaining scenarios, Neighbour Replication and Multi-Publication's results are affected by the loss of keys; these are discussed further ahead in the text. For $c = 1$, it is clear that Rollerchain transfers significantly less objects than the other replication

Table 1. Number of objects moved per node vs Percentage of objects reachable at the end of the simulations

	$c = 1$	$c = 10$	$c = 100$
Rollerchain	38.0/ 100.0%	40.2/ 100.0%	44.6/ 98.2%
Neighbour Replication	77.0/ 100.0%	93.0/ 94.1%	0.85/ 0.0%
Multi-Publication	88.5/ 100.0%	31.3/ 10.2%	0.22/ 0.0%

techniques. This happens because in Rollerchain most data transfers are useful, i.e., they are used to recover the target replication degree. In the competing approaches, many data transfers have no impact on reliability, they are performed exclusively to preserve some replication-topological constraint (such as to keep replicas close to the key owner) with no benefit on data availability.

When a node joins Chord, it must receive all the objects it will own and one replica of each object its $R - 1$ neighbours own. Being O_t the total number of objects stored in the system and N the size of the network, on average each node join will cause the transfer of $(O_t/N) \times R$ objects. Conversely, a node failure triggers the creation of $(O_t/N) \times R$ new replicas. The combined effect of a failure followed by a join causes $2 \times (O_t/N) \times R$ data transfers. Rollerchain avoids creating new copies of data immediately after a failure. By construction, joins happen in vnodes that have lost members and replicas are created to restore the replication degree at that time.

In any of the schemes, the minimum number of objects transferred per joining node would be $(O_t/N) \times R$ (i.e., 35 in our setting). Rollerchain's results are close to this value, whereas Neighbour Replication's results are closer to 70, which matches the expected results as discussed above. Naturally, none of the architectures achieves the theoretical minimum given that under churn, transient routing inconsistencies lead to data being replicated at wrong locations (and later discarded). This effect is particularly noticeable for Multi-Publication, as this scheme relies heavily on overlay routing to perform replication.

We now discuss the results for higher churn rates. Frequent changes in the network topology cause havoc in the replica maintenance schemes of Neighbour Replication and Multi-Publication, due to the increasing number of routing inconsistencies. In fact, Multi-Publication loses a large number of objects during the simulation, for $c = 10$. This effect is also observed for Neighbour Replication, for $c = 100$. Since objects are lost, the number of moved objects decreases (obviously, the number of moved objects in this case can no longer be compared with Rollerchain, where most objects are still reachable in these conditions). Rollerchain is much more robust, as the DHT topology remains stable. Still, under heavy churn, even in Rollerchain, new nodes become unable to perfectly replace the failed ones, inducing occasional changes in the structured layer through divisions and merges of virtual nodes, some minimal data loss (less than 2%), and some performance degradation.

Overlay maintenance costs

Figures 3 and 5 show the bandwidth consumption resulting from the two remaining sources of overhead: *i*) the signalling costs of the monitoring protocols that are required to preserve the replication degree (Figure 3); and *ii*) the costs associated with maintaining the overlay topology (Figure 5).

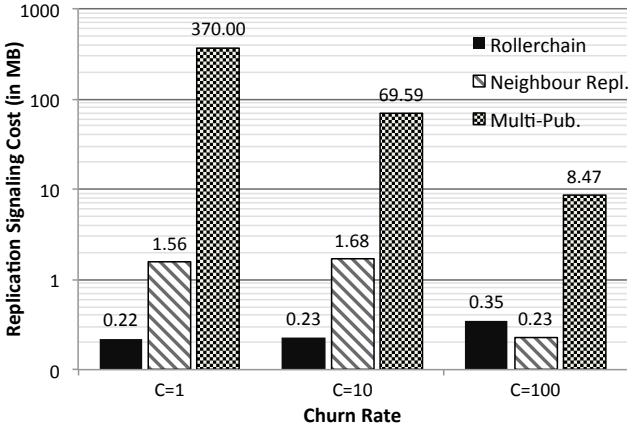

Fig. 3. Replication signalling costs per node, considering 128bit keys.

The costs of maintaining replication in Rollerchain are not significantly different from those of Neighbour Replication. In both protocols, each node has to gossip with a limited number of neighbours (one in the case of Rollerchain, as each pnode gossips with only one other neighbour on each round, and $R-1$ in the case of Neighbour Replication). Also, in both cases, the message contains only a simple hash of the node's data. Multi-Publication's costs are at least two orders of magnitude higher. This results from the periodic lookups that a node is required to perform for every key it owns. These lookups are routed in the overlay and incur in multiple message exchanges. In fact, when considering only the periodic replication mechanisms, Multi-Publication requires a node to monitor $R-1$ nodes in the overlay for each key it stores. Since the replica nodes for each object are distributed in the overlay using consistent hashing, the larger the system is the less likely it is that different objects are replicated in the same node. Thus, even in stable topologies (with no node joins or leaves), the number of neighbours each node has to contact increases with the network size, the number of objects, and their replication degree. This property severely limits the scalability of Multi-Publication (as Figure 4 shows).

When considering the topology maintenance overhead (Figure 5), it can be observed that Rollerchain is more expensive than the other two techniques. Such results are not surprising, as Rollerchain combines design elements of two overlays. Still, these costs are in the same order of magnitude of competing approaches, and relatively small when compared with the costs of replication discussed above. For example, considering a churn rate of $c = 1$, if objects have

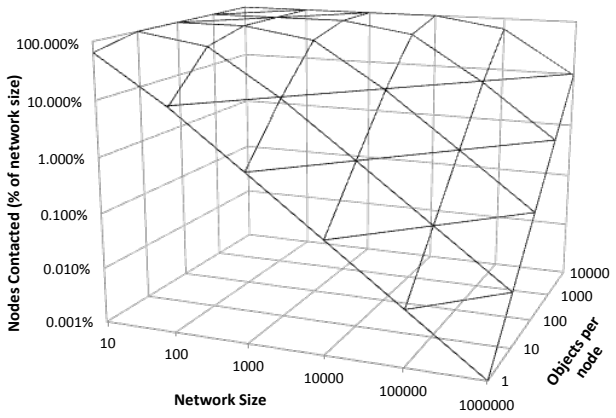

Fig. 4. Average number of nodes contacted by nodes running multi-publication, considering $R = 10$.

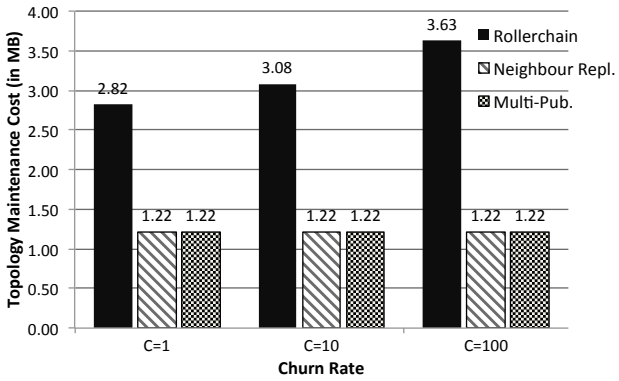

Fig. 5. Topology signalling costs per node.

a size of 5MB (the typical size for audio files), each node in Rollerchain will require 190MB of bandwidth to move objects; whereas each node using Neighbour Replication would consume approximately 385MB of bandwidth. In fact, for a system with the number of objects and replication degree considered in this section, the overall costs of Rollerchain are below those of Neighbour Replication for object sizes greater than 6.7KB. Finally, the increase in bandwidth consumption by Rollerchain under higher churn rates can be explained by the occurrence of merge and division operations, which require additional coordination among peers, but also make the operation of the overlay more robust as discussed earlier.

Table 2. Statistics for number of queries answered per node. Table presents average (AVG), minimum (MIN), maximum (MAX) and standard deviation (STDDEV) for Uniform (U) and Zipf (Z) distributions.

		MIN	AVG	MAX	STDDEV(K)
U	Rollerchain	212	500	739	116
	Neighbour Repl.	22	500	1907	241
	Multi-Publication	37	500	1684	203
Z	Rollerchain	210	500	0.739×10^3	116
	Neighbour Repl.	0	500	1774×10^3	20782
	Multi-Publication	0	500	444×10^3	4496

5.3 Load Balancing

To evaluate the load balancing capacity of Rollerchain, we have conducted experiments with stable topologies (*i.e.*, while no join or leave operations occurred). This decouples the load balancing properties from other aspects of the overlay operation. We assigned keys to 50.000 objects following two distributions: *i*) uniform and *ii*) Zipf with a 1.25 skew. The uniform random distribution provides a baseline of comparison, while the Zipf distribution illustrates operation in scenarios where a portion of the identifier space becomes overloaded. Subsequently, 100 queries per object were triggered at random nodes to achieve a grand total of 5.000.000 queries.

Table 2 presents the resulting number of queries answered by each individual peer in the overlay. One can observe that the variance of the results obtained with Rollerchain is smaller than with the competing strategies. In Rollerchain, most nodes have approximately the same load. In fact, even the highest loaded node in Rollerchain only has 50% more load than the average; with Neighbour Replication, this value rises to 280%.

We now compare the results obtained with the Uniform (U) and the Zipf (Z) key distributions. We highlight that the results obtained for Rollerchain in the Zipf scenario are very similar to those obtained in the Uniform scenario. This shows that Rollerchain's dynamic partitioning of the data can adapt to extremely skewed datasets, making it oblivious to the key distribution in the identifier space. This makes Rollerchain suitable for supporting applications that use non-uniform distributions of keys. Multi-Publication performs better than Neighbour Replication under the Zipf distribution. However, due to the lack of mechanisms to adapt the distribution of objects among nodes, the skew in the load imposed across nodes is, at least, one order of magnitude above that achieved by Rollerchain.

6 Conclusions

This paper proposes a novel combination of gossip-based mechanisms and structured overlays to generate a DHT of virtual nodes, where each virtual node is

materialized by a set of physical nodes. The resulting system can save a significant amount of bandwidth consumption when maintaining replication under churn: Our solution, named Rollerchain, has proved experimentally to be more robust than competing approaches. As future work, we plan to explore new data persistence models to apply in Rollerchain, allowing applications to specify different replication requirements for each particular object.

Acknowledgments: This work was partially supported by FCT project "HPC-LSI" (PTDC/EIA/102212/2008) and via the INESC-ID multi-annual funding through the PIDDAC Program fund grant, under project PEst-OE/EEI/LA0021/2013. The authors wish to thank F. Araújo, M. Couceiro, F. Cristovão, H. Miranda, J. Pereira, and P. Romano their comments.

References

1. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc of SIGCOMM '01*, San Diego (CA), USA, 2001.
2. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *In proc. of Middleware '01*, Heidelberg, Germany, 2001, pp. 329–350.
3. P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. of the 1st IPTPS*, Cambridge (MA), USA, 2002.
4. I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse, "Kelips: Building an efficient and stable p2p DHT through increased memory and background overhead," in *Proc. of the 2nd IPTPS*, Berkeley (CA), USA, 2003.
5. J. Alveirinho, J. Paiva, J. Leitão, and L. Rodrigues, "Flexible and efficient resource location in large-scale systems," in *Proc. of the 4th LADIS*, Zürich, Switzerland, July 2010, pp. 55–60.
6. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Comm.*, vol. 20, no. 8, 2002.
7. S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *Proc. of the 11th NOSSDAV*, Port Jefferson (NY), United States, 2001.
8. F. Dabek, M. K. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *Proc. of the 8th SOSP*, Banff, Alberta, Canada, 2001, pp. 202–215.
9. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc of SIGCOMM*, San Diego (CA), United States, 2001.
10. P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *Proc. of the 8th HOTOS*, Elmau/Oberbayern, Germany, 2001, p. 75.
11. D. Wu, Y. Tian, and K.-W. Ng, "Analytical study on improving dht lookup performance under churn," in *Proc. of the 6th P2P*, Cambridge, UK, 2006.
12. C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Lihue, Hawaii, May 2003, pp. 1–6.

13. P. Knezevic, A. Wombacher, and T. Risse, "Enabling high data availability in a DHT," in *The 16th Int'l Workshop on Database and Expert Systems Applications (DEXA)*, Copenhagen, Denmark, August 2005.
14. M. Waldvogel, P. Hurley, and D. Bauer, "Dynamic replica management in distributed hash tables," in *Research Report RZ3502, IBM*, 2003.
15. P. Knezevic, A. Wombacher, and T. Risse, "DHT-based self-adapting replication protocol for achieving high data availability," in *Proc. of SITIS'06*, Hammamet, Tunisia, 2006, pp. 201–210.
16. S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, "Performance evaluation of replication strategies in dhts under churn," in *Proc. of the 6th MUM*, Oulu, Finland, 2007, pp. 90–97.
17. L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Scalable consistency in scatter," in *Proc. of the 23rd SOSP*, Cascais, Portugal, 2011, pp. 15–28.
18. L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, pp. 133–169, May 1998.
19. A. Ghodsi, S. Haridi, and H. Weatherspoon, "Exploiting the synergy between gossiping and structured overlays," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 61–66, 2007.
20. E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro, "Peercube: A hypercube-based p2p overlay robust against collusion and churn," in *Proc. of the 2nd SASO*, Venice, Italy, Oct. 2008.
21. H. Ribeiro and E. Anceaume, "Datacube: A P2P persistent data storage architecture based on hybrid redundancy schema," in *Proc. of the 18th Euromicro Int'l Conf. on Parallel, Distributed and Network-Based Computing*, Pisa, Italy, Feb. 2010.
22. H. Abu-Libdeh, H. Geng, and R. van Renesse, "Elastic replication for scalable consistent service," in *SOSP (extended abstract)*, Cascais, Portugal, 2011.
23. B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *Proc. of the 23rd INFOCOM*, March 2004, pp. 2253 – 2262.
24. K. Kenthapadi and G. S. Manku, "Decentralized algorithms using both local and random probes for P2P load balancing," in *Proc. of the 7th SPAA*, Las Vegas, Nevada, USA, 2005, pp. 135–144.
25. J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity and churn," in *Proc. of the 24th INFOCOM*, Miami, USA, 2005.
26. M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "The peer sampling service: experimental evaluation of unstructured gossip-based implementations," in *Proc. of Middleware*, Toronto, Canada, Oct. 2004.
27. B. Maniymaran, M. Bertier, and A.-M. Kermarrec, "Build one, get one free: Leveraging the coexistence of multiple P2P overlay networks," in *Proc. of the 27th ICDCS*, Toronto, Canada, 2007.
28. J. Paiva, J. Leitão, and L. Rodrigues, "Rollerchain: a DHT for high availability," in *The 11th ACM/IFIP/Usenix Middleware Conference (Poster Session)*, Lisboa, Portugal, Dec. 2011.
29. T. Shafaat, B. Ahmad, and S. Haridi, "Id-replication for structured peer-to-peer systems," in *Proc. of the Euro-Par*, Rhodes Island, Greece, Aug. 2012.
30. J. Leitão and L. Rodrigues, "Overnesia: a robust overlay network for virtual super-peers," INESC-ID, Tech. Rep. 36, July 2009.
31. D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in *Proc. of the 21st PODC*, Monterey, California, 2002.

32. N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: a scalable overlay network with practical locality properties," in *Proc. of 4th USITS*, Seattle, WA, 2003.
33. F. Klemm, S. Girdzijauskas, J.-Y. Le Boudec, and K. Aberer, "On routing in distributed hash tables," in *Proc. of the 7th P2P*, 2007.
34. A. Ghodsi, L. O. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," in *Proc of the DBISP2P*, Trondheim, Norway, 2007.
35. M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.