ID-Replication for Structured Peer-to-Peer Systems*

Tallat M. Shafaat[1], Bilal Ahmad[1], and Seif Haridi[2]

[1] KTH - Royal Institute of Technology, Sweden
[2] Swedish Institute of Computer Science, Sweden
{tallat,bilala,haridi}@kth.se

Abstract. Structured overlay networks, like any distributed system, use replication to avoid losing data in the presence of failures. In this paper, we discuss the short-comings of existing replication schemes and propose a technique for replication, called *ID-Replication*. ID-Replication allows different replication degrees for keys in the system, thus allowing popular data to have more copies. We discuss how ID-Replication is less sensitive to churn compared to existing replication schemes, which makes ID-Replication better suited for building consistent services on top of overlays compared to other schemes. Furthermore, we show why ID-Replication is simpler to load-balance and more secure compared to successor-list replication. We evaluate our scheme in detail, and compare it with successor-list replication.

1 Introduction

Structured overlay networks provide the infrastructure used to build scalable and fault-tolerant key-value stores, e.g. Cassandra [9]. While scalability comes with using consistent hashing, fault-tolerance is achieved by replication. There are different strategies for replication in overlays, such as successor-list replication [17], using multiple hash functions, and symmetric replication [3]. Out of these, successor-list replication is the most popular and widely used in ring-based overlays. For instance, overlays including Chord [17], Pastry [14] (with a minor modification), and Cassandra [9], all use successor-list replication.

It turns out that successor-list (SL) replication has some drawbacks. SL-replication is highly sensitive to churn; hence a single node join or failure event results in updating multiple replication groups. Furthermore, the replication degree has to be constant throughout the system, restricting popular/hot data from having more replicas. Next, SL-replication is inherently difficult to load-balance. Finally, SL-replication is less secure and presents a bottleneck since there is a master replica of each replication group and all requests for that group have to go through the master replica. We discuss these issues in detail in Section 2.1.

In this paper, we propose a replication strategy called *ID-Replication*. ID-Replication does not suffer from the afore-mentioned drawbacks of SL-replication.

* We would like to thank Cosmin Arad, Ahmad Al-Shistawy and Niklas Ekström for their valuable discussions and feedback.

C. Kaklamanis et al. (Eds.): Euro-Par 2012, LNCS 7484, pp. 364–376, 2012.

It allows varied replication degrees in the system, and requests do not need to go through the master replica. ID-Replication gives more control to an administrator, without hampering self-management. Furthermore, ID-Replication is less sensitive to churn, thus being better suited to be used for building consistent services and in asynchronous networks where false failure detections are a norm. Since we use a generic design, ID-Replication can be used in any structured overlay network.

In this paper, we discuss the short-comings of popular existing replication schemes. We explain ID-Replication in detail and discuss the ideology behind the design decisions. We perform a thorough evaluation and compare ID-Replication to SL-replication.

2 Preliminaries

An overlay makes use of an *identifier space*, which for our purposes is defined as a set of integers $\{0, 1, \cdots, \mathcal{N}-1\}$, where $\mathcal{N}$ is some apriori fixed, large, and globally known integer. This identifier space is perceived as a ring that wraps around at $\mathcal{N}-1$. Each node in the system has a unique identifier from the identifier space. The *successor* of a node with identifier p is the first node found going in clockwise direction on the ring starting at p. Similarly, the *predecessor* of a node with identifier q is the first node met going in anti-clockwise direction on the ring starting at q. The *successor-list* of a node m consists of m's c immediate successors, where c is typically set to $\log_2(s)$, where s is the network size.

Each node q is responsible for storing keys between q's predecessor and q. For a replication degree of r in SL-replication, a key k is stored on the node q that is responsible for storing k, and $r-1$ immediate successors of q. In essence, the key is stored on the responsible node q, and the first $r-1$ members of q's successor-list (see Figure 1). In Fig 1, node 30 is responsible for storing keys $k \in (20, 30]$, and k are replicated on $\{30, 35, 40\}$, which is called the *replica group* for k. As nodes join and leave the system, the successor, predecessor and successor-lists are updated, leading to changes in the replica groups and transfer of keys between nodes.

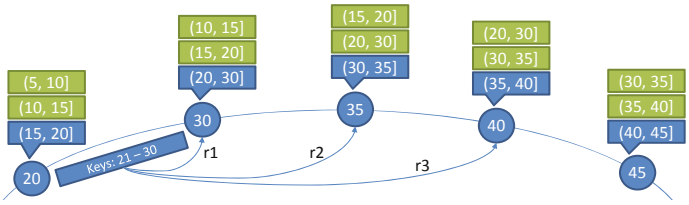

Fig. 1. Successor-list replication with replication degree 3. The replication group for *keys* $\in [21, 30]$ is $\{30, 35, 40\}$. Similarly, responsibility of node 35, i.e. $(30, 35]$, is replicated on 3 nodes encountered clockwise from 35, i.e. 35, 40 and 45.

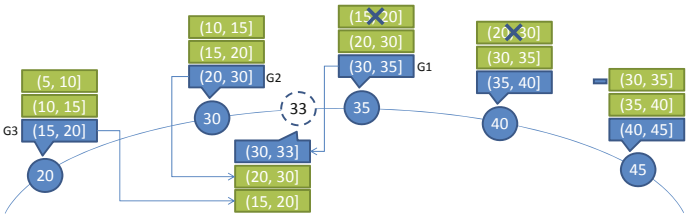

Fig. 2. A new node 33 joins in a system using successor-list replication and degree 3. 6 nodes are involved in making changes, and 4 replication groups have to be updated.

2.1 Problems with Existing Schemes

Replica Groups Affected by Churn: Churn - node joins and failures - is considered a norm in P2P systems. A desirable behaviour is that a churn event should not effect the configuration of an overlay greatly. In SL-replication, the unit of replication is a node's assigned key space, also known as the node's responsibility. For instance in Fig. 1, the key space assigned to node 30 is $(20, 30]$, which is replicated on 35 and 40. Consequently, for a replication degree of r, each node replicates r node responsibilities going anti-clockwise.

When a new node joins the overlay, it divides a node responsibility range into two ranges. Similarly, a node failure results in merger of two node responsibilities. Since each node responsibility range is replicated on r nodes, and each node replicates r ranges, a single churn event results in reconfiguration of r replication groups. Furthermore, a join event involves action on behalf of $2 \times r$ nodes, and a failure involves action on behalf of $(2 \times r) - 1$ nodes. This is shown in Figure 2 where a new node 33 joins the system in a state shown in Figure 1. Replication groups $G1$, $G2$, and $G3$ need to be updated, and nodes 20, 30, 33, 35, 40, 45 are involved in such updates.

This approach has multiple drawbacks. First, a single churn event is overly complicated, involving many nodes. Second, consistent services built on top of overlays require consistent views of replication groups. For instance, Scatter [4] and Etna [12], both require consensus whenever a replication group changes. A high number of reconfigurations for a single churn event is undesirable. Lastly, the time duration needed to stabilize for a single churn event is very high.

Load-Balancing: We argue that SL-replication is complicated to load-balance. Consider an unbalanced system, such as the one depicted in Figure 1. It is unbalanced in-terms of keys since node 30 is storing 10 keys while all other nodes are storing 5 keys. A simple load-balancing mechanism, such as [8], would move node 30 counter-clockwise to handover responsibility of some keys to 35, or move 20 clockwise so that 20 takes over responsibility of some keys from 30. Since $keys \in (20, 30]$ are replicated on 3 nodes, such a movement will reduce load from one replica node only. Hence, r node movements on the identifier ring are needed to balance the load of one key range.

Security: In SL-replication, all requests for a key k end up on the node n responsible for k. This has two drawbacks. First, it is difficult to load-balance requests since all requests for k pass through n before they can be routed to a replica. Hence, n becomes a bottleneck. Second, if n is an adversary, it can launch a malicious attack [16].

Symmetric Replication: In Symmetric Replication [3], keys are stored symmetrically on the identifier space using equivalence classes. This leads to requiring a complicated bulk operation for retrieving all keys in a given range. Node joins and failures have to use such a bulk operation to find data to be replicated.

3 ID-Replication

In this section, we describe a replication scheme for ring-based overlays, called ID-Replication. We first provide an overview of ID-Replication, give a detailed algorithmic specification, and then discuss its desirable properties.

3.1 Overview

We set out to design a replication scheme that is less sensitive to churn in terms of the number of replication groups that need to be reconfigured. In ID-Replication, we use sets of nodes, called *groups*, instead of individual nodes as the building blocks for the overlay. Instead of partitioning the identifier space amongst nodes, we partition the identifier space among groups. Thus, compared to the simple structured overlay model where nodes are responsible for key ranges, we assign responsibility ranges to groups. Consequently, groups are assigned identifiers. The idea of using groups instead of nodes can be applied to the majority of the overlays. For the sake of simplicity, we use Chord-like notation in this paper.

All nodes within a group have the same identifier as the group. To distinguish nodes within a group, each node also has a group-local identifier. The group-local identifiers of nodes only need to be unique within the group. For efficient routing, each node maintains long range links, such as fingers in Chord.

The model of ID-Replication is shown in Figure 3. There are five groups on the identifier space: $20, 30, 35, 40$ and 45. The successor of a group is the first group encountered going clockwise from that group, e.g. group 40 is the successor of group 35. Similarly, the predecessor of a group is the first group encountered going anti-clockwise, e.g. 30 is the predecessor of 35. A group is responsible for the key range from its predecessor to itself, e.g. group 35 is responsible for $keys \in (30, 35]$.

Each group is composed of a number of nodes, e.g. group 30 contains nodes $\{1, 2, 3\}$. The nodes of a group are the replicas for the keys that the group is responsible for. The size of each group is specified using two parameters: r_{min} and r_{max}. Thus, the replication degree of keys is always between r_{min} and r_{max}.

To maintain the ring under dynamism, we employ a modified version of periodic stabilization [17] that operates on groups instead of nodes. Furthermore,

we use *gossiping* between nodes in a group to synchronize the view of the group among the group members.

We use two operations for reconfiguring groups: *Merge* and *Split*. When the size of a group $G1$ drops below r_{min}, we need to *merge* $G1$'s members with another group $G2$ such that the size of the merged group should be less than r_{max}. The merged group $G = G1 \cup G2$ retains the identifier of $G2$.

When the size of a group G becomes larger than r_{max}, we need to *split* it into two groups, $G1$ and $G2$, such that the size of each split group is larger than or equal to r_{min}. The identifiers of $G1$ and $G2$ are calculated in a way to increase the load-balance in the system.

A failure of a node can trigger a merge. Similarly, a new node joins an existing group, which can result in a split.

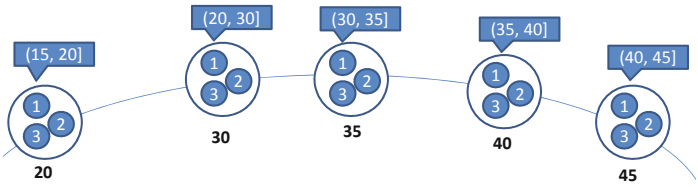

Fig. 3. A configuration of ID-Replication. Replica groups are denoted by a single identifier on the identifier space ring. Nodes in a replica group $G1$ are responsible for storing keys between $G1$'s predecessor replica group's identifier and $G1$. Nodes within a replica group are differentiated by using group-local identifiers (1, 2, and 3 in the figure).

3.2 Algorithm

We give a full specification of ID-Replication as Algorithm 1 and 2 in an *rpc*-notation. Each node stores a group-local identifier l_{id}, a group identifier id, and a set of nodes in its group, *group*. An rpc-call is denoted by '::'. For instance, `m::id` denotes the value of id on node m.

A new node n joins the system by attempting to become a member of a group of size less than r_{max} to avoid a split operation. Ideally, n should join the lowest-sized group. Such a group can be found in a best-effort manner by a random walk, or by maintaining directories that store such information (as in [5]). If a group with less than r_{max} members is not found, n will join a group causing it to split into two.

Nodes maintain successor-lists to preserve the ring-geometry amid churn. The difference between Chord and ID-Replication successor-lists is that the lists are composed of successive groups instead of successive nodes. If all nodes in the successor group of G fail or merge with another group, G points to the next group in the successor-list (Algo 1, line 13). For ring and successor-lists maintenance, we use an algorithm similar to Chord's periodic stabilization, where nodes belonging to a group periodically stabilize the ring with nodes in their successor group (Algo 1, lines 15–27).

Algorithm 1. ID-Replication(part 1): Periodic stabilization for joins and failures

```
                                                  ▷ '::' denotes a remote procedure call
 1: n.join(seed)                    ▷ Periodically retried with new seed if request fails
 2:     seed::join_request(n)
 3:
 4: n.join_request(m)
 5:     if |group| < r_max then                          ▷ if n's replica group has space
 6:         group := group ∪ {m}
 7:         m :: ⟨id, group⟩ := ⟨id, group⟩          ▷ Set joining node's id and group
 8:
 9: n.node_failure(f)                                              ▷ Node f failed
10:     group := group − {f}
11:     pred.group := pred.group − {f}
12:     succ.group := succ.group − {f}
13:     if succ.group == {} then
14:         succ := next_in_successor_groups()
15:
    ▷ Periodically check for new successor and predecessor groups
16: n.stabilize_ring()
17:     random_succ := select_random(succ.group, 1)          ▷ Select a random node
18:     ⟨x.id, x.group⟩ := rand_succ :: pred.⟨id, group⟩
19:     if x.id ∈ (id, succ.id) then
20:         ⟨succ.id, succ.group⟩ := ⟨x.id, x.group⟩
21:     succ.group := select_random(succ.group, 1):: group          ▷ Update my view
22:     ∀p ∈ succ.group do p :: notify(n, id, group)
23:
24: n.notify(src, pid, pgroup)
25:     if pid ∈ (pred.id, id) or  src ∈ pred.group then
26:         ⟨pred.id, pred.group⟩ := ⟨pid, pgroup⟩
27:
```

Say the size of a group G_{size} is more than r_{min}. In such a case, even if $G_{size} - r_{min}$ nodes, called *standby* nodes, leave the group, it will neither violate the replication degree nor require a merge operation. These standby nodes can potentially become part of a group in which a node fails. Hence, standby nodes advertise themselves (Algo 2, lines 16–17) by either gossiping, or periodically updating their address information into directories (as in [5]).

Each node n periodically checks if the size of its group, G_{size}, is between r_{min} and r_{max}. If G_{size} is smaller than r_{min}, then n searches for a standby node by gossiping or contacting a directory, and tries to include it in n's group. If a standby node cannot be found, n triggers a merge operation (Algo 2, lines 7–14). A merger is required in this case to maintain a replication degree of at least r_{min}. Similarly, if G_{size} is larger than r_{max}, n initiates the split operation by dividing the group into two groups (Algo 2, lines 1–6). Furthermore, n periodically gossips with its group members to synchronize their view of the group, and can use anti-entropy to update data items.

Algorithm 2. ID-Replication(part 2): SPLIT and MERGE operations

▷ Periodically attempt to keep $r_{min} < |group| < r_{max}$
1: **every** γ **time units and** $|group| > r_{max}$ **at** n ▷ SPLIT operation
2: $peers\_to\_split$:= get_top(sort($group$), r_{min}) ▷ Get r_{min} nodes with lowest l_{id}
3: ∀ p ∈ $peers\_to\_split$ **do** p::$\langle id, group\rangle := \langle new\_key, peers\_to\_split\rangle$
4: $peers\_to\_retain := group - peers\_to\_split$
5: ∀ p ∈ $peers\_to\_retain$ **do** p::$\langle id, group\rangle := \langle id, peers\_to\_retain\rangle$
6: **end event**

7: **every** γ **time units and** $|group| < r_{min}$ **at** n ▷ Due to failures
8: $node$:= search_standby_node()
9: **if** $node = nil$ **then** ▷ Search failed, MERGE with successor group
10: $\langle new\_id, new\_group, new\_succ\rangle := \langle succ.id, succ.group \cup group, succ :: succ\rangle$
11: ∀ p ∈ new_group **do** p::$\langle id, group, succ\rangle := \langle new\_id, new\_group, new\_succ\rangle$
12: **else** ▷ Make the standby node part of n's group
13: node::$\langle id, group, succ, pred\rangle := \langle id, group \cup \{node\}, succ, pred\rangle$
14: **end event**

15: **every** δ **time units at** n ▷ Periodically synch view with group-mates
16: **if** $index\_of$(n, $sort(group)$) $> r_{min}$ **then**
17: publish_as_standby_node(n)
18: gossip_view($group$) ▷ Synchronize group view (& data) with group members
19: **end event**

3.3 Discussion

As we discuss in Section 3.1 and evaluate in Section 4, ID-Replication requires less replication group reconfigurations per churn event. This makes ID-Replication ideal for building a consistent DHT. Each replication group can be considered as a replicated state machine and operations are performed on the data in a total order within the group. To handle dynamism, we need to support the merge and split operations where the view of a group changes. For this, we can use a reconfigurable replicated state machine, such as SMART [11]. Using SMART with SL-replication is both complicated and expensive as replicated state machines are implemented using Consensus. Since ID-Replication requires fewer replication group reconfigurations per churn event, it will require fewer instances of consensus. Furthermore, in an asynchronous system, false failure suspicions are very common, which will trigger much more reconfiguration requests in SL-replication.

ID-Replication allows the system user to have different replication degrees for different keys. We use two parameters, r_{min} and r_{max}, to control the replication degree. For a given range, the number of replicas is at least r_{min} and at most r_{max}. Thus, popular or critical data can have more copies than other data by setting higher values of r_{min} and r_{max} for the corresponding key range. Furthermore, requests do not need to go through a master replica. Hence, ID-Replication does not have any bottlenecks, and requests can be load-balanced

across all replicas. Finally, such a design avoids the security vulnerabilities of SL-replication [16].

Owing to the design of ID-Replication, a system administrator has much more control over the system compared to SL-replication. For instance, the administrator can control how many and which machines should serve a particular key-range. This also allows the usage of specialized hardware for handling requests for certain keys. On the contrary, a node in SL-replication is responsible for replicating multiple key ranges (r key partitions anti-clockwise), making it harder to control.

Routing tables, e.g. fingers in Chord, can also be build using groups. Each routing pointer can point to a group, containing addresses of multiple nodes. Greedy routing can be done on group identifiers, and a lookup can be routed to a random node in the group. For fault-tolerance and better performance, a lookup can be routed by forwarding in parallel to all nodes in the groups at each hop, and considering only the first reply. While such a mechanism consumes more bandwidth, it (a) is more reliable as it can tolerate failure of nodes in the path, and (b) has lower latency as the lookup can exploit multiple paths. Such parallel lookup techniques have also been proposed for Chord like overlays [10].

4 Evaluation

To evaluate our work, we simulated both ID-Replication and SL-replication in Kompics [2]. The simulations were performed with an initial network size of 2000 nodes, using the King dataset [6] for message latencies between the nodes. Each experiment had the following structure: we initialized the overlay with 2000 nodes. Once the overlay converged, we subjected it to 2000 churn events (1000 joins and 1000 failures), and measured the metrics till the topology converged. The lifetimes of nodes had a poisson distribution, and each node failure was followed by a join event. We evaluated both replication schemes under various levels of churn by changing the median parameter of poisson distribution for the lifetimes. A higher median lifetime results in lower churn rate. We performed simulations for periodic stabilization periods of 30 and 60 seconds. The experiment results for both stabilization rates were the same, so we omit graphs for stabilization delay of 60 seconds due to space restrictions. We simulated 3 directories for nodes to publish and find standby nodes, and used a value of $r_{max} = 2 \times r_{min}$. Such directories can be implemented by using predefined keys, and storing information under those keys [5]. We repeated each experiment for 10 seeds and report the averages.

4.1 Replication Groups Restructured

We measured the number of replication groups that need to be reconfigured due to the churn events (see Figure 4). The x-axis shows the median lifetime used for nodes, while the y-axis depicts the number of replication groups restructured per churn event. As analyzed earlier, the figure shows that there are r number

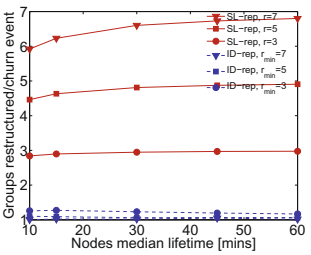

Fig. 4. Number of replication groups restructured per churn event

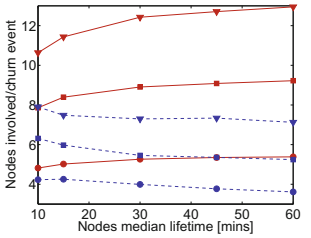

Fig. 5. Number of nodes involved in updates for each churn event

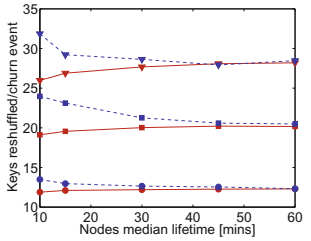

Fig. 6. Number of keys transferred between nodes converge for both schemes

of reconfigurations per churn event for SL-replication, while the corresponding value stays close to one for ID-Replication. In this case, ID-Replication does not depend on the replication degree whereas SL-replication does. SL-replication has lower restructuring count at higher churn rates than lower rates. The reason being that at high churn, simultaneous node failures in a replica group can mask the cost of multiple node failures with the cost of a single node failure, since, all the failed nodes will be replaced in a single periodic stabilization round.

ID-Replication has a very low group restructuring cost and is unaffected by both r and churn rate because majority of churn events restructure only one group. Splits, merges and standby node movement restructure two groups. Since these events occur at a low frequency, the restructuring cost stays low.

4.2 Nodes Involved in Updates

Each churn event requires action on behalf of a certain number of nodes. In this experiment, we counted the number of nodes involved in the reconfiguration updates. This count is depicted in Figure 5, normalised against the number of churn events. As analyzed in Section 2.1, the count for SL-replication approaches $2 \times r$. Since ID-Replication involves only one group for a single churn event (excluding splits and merges), the number of nodes involved to handle churn stays close to r. It is noteworthy that the performance of ID-Replication improves as the mean life time of nodes increases, which is opposite to SL-replication behaviour. At lower churn rates, the number of splits and merges is reduced because of the join mechanism of ID-Replication where new nodes try to join groups with low node count. Since at low churn rates the topology changes very slowly, nodes take better decisions about which group to join. This reduces splits and merges, thus resulting in fewer nodes involved at low churn rates. Such behaviour makes ID-Replication ideal for managed systems in data-centers and cloud computing where the churn rate is low.

4.3 Keys Reshuffled

Next, we evaluate the number of keys that have to be transferred between nodes. Figure 6 shows the comparison between SL-replication and ID-Replication with-respect-to the number of keys re-shuffled per churn event. At lower churn rates, both SL-replication and ID-Replication converge to the same value. However, the two replication schemes behave differently at higher churn rates.

The reason behind SL-replication's reduced cost at high churn rate is because nodes become aware of the change in their responsibilities after each periodic stabilization step. Now, when the mean lifetime of nodes is very short it may happen that a failed node is replaced by a new node, before other nodes had the chance to detect its failure. This way the join event masks the cost associated with the failure of the node. Furthermore, a new node may fail shortly after joining (within a periodic stabilization window) without anyone noticing its join and failure, thus avoiding key re-shuffling.

The increased cost of ID-Replication at high churn is due to a high merge rate. Merge is costly in terms of keys re-shuffled as it results in transferring keys of two responsibility ranges by all members of the two groups being merged. On the other hand, the movement of a standby node requires the transfer of one responsibility range only once. When the churn rate is higher than the rate at which the standby nodes are being advertized, the number of merge operations is naturally higher. However, when the churn rate becomes comparable to the rate of publication of standby nodes, the system involves more standby node movements and thus reducing the number of merge operations. This phenomena is depicted in Figure 7, illustrating the number of splits, merges, and standby node movements per churn event. The figure shows that as the mean lifetime of nodes is increased (churn rate is reduced), the rate of standby node movements increases, which results in a decreased merge rate. Furthermore, at low churn rates, the search for lowest size group gets better results. This experiment suggests that for higher rates of churn, the rate of updating the directories with group-size and standby node information should be higher as well. It is worth noting that a lifetime of 10 minutes is considered a very high churn rate for a DHT.

4.4 Overhead of Maintaining Groups

ID-Replication maintains groups using a gossiping protocol such as Cyclon [18], which adds to the maintenance cost. Cyclon is inexpensive, especially given that the group sizes are small and the churn rates are moderate in cloud environments. We used a gossip rate equal to the periodic stabilization rate (30 seconds), and measured maintenance cost for various network sizes. Our results show that the gossiping overhead is approximately the same as periodic stabilization. Hence, using ID-Replication doubles the maintenance bandwidth requirement and the number of messages exchanged is almost 1.7 times higher. The maintenance cost is still moderate and negligible given today's interconnects.

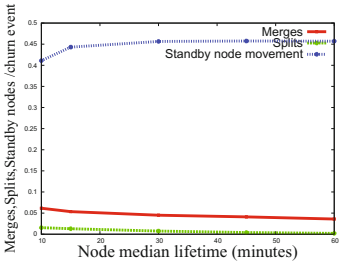

Fig. 7. The number of standby node movements increases with decreasing churn rate, thus reducing group merges

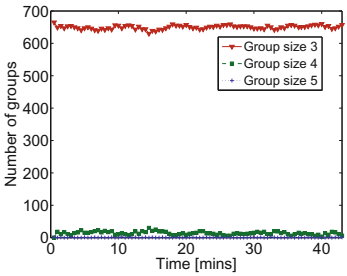

Fig. 8. Number of groups for sizes between 3 (r_{min}) and 6 (r_{max})

4.5 Evolution of Groups

Finally, we evaluated the size of groups over time, where it is desirable that the group sizes are close to r_{min}. Figure 8 shows the number of groups for each size between r_{min} and r_{max} over time for a poisson churn with mean 60 minutes. The figure confirms that most of the groups have a size of 3, which is r_{min}. We observed similar trends for other churn rates as well, which we omit here due to space constraints.

5 Related Work

Symmetric replication [3] proposes an alternative replication scheme for structured overlays. However, it requires a bulk search operation to find all data items in a key range for every join and fail event. Such a bulk operation is complex, requires extra messages, and induces a delay before the churn event is completely catered. In contrast, we do not require any such bulk operation.

Scatter [4] uses a similar scheme for achieving consistency in DHTs. Compared to our scheme, they further sub-divide the groups to differentiate between key responsibilities of each node. Furthermore, they do not evaluate or argue for the usefulness of their scheme. We provide algorithmic specification of our work, backed by design decisions and evaluation with comparison to SL-replication.

P-Grid [1] uses a notion called *structural replication*, where nodes form groups and data is replicated among nodes in these groups. Like ID-Replication, different groups can have different replication degrees. The geometry of P-Grid is a tree, while we give a solution for overlays with a ring geometry, which is the geometry of a majority of structured overlays. Compared to P-Grid, our solution uses consistent hashing [7], thus leveraging properties of consistent hashing such as self-management, load balancing, and minimized repartitioning of data under churn.

Agyaat [15] proposes to use groups of nodes, called *clouds*, to provide mutual anonymity in structured overlays. Compared to ID-replication, Agyaat maintains an R-Ring and an overlay with the clouds, which is more complicated and

requires some nodes to be part of two overlays. A similar approach is taken by Narendula et al. [13], where nodes form sub-overlays with trusted nodes for better access control in P2P data management.

6 Conclusion

This paper discusses popular approaches employed for replication in structured overlay networks, including successor-list replication and symmetric replication, and outlines their drawbacks. We present the design, algorithmic specification, and evaluation of ID-Replication, a replication scheme for structured overlays that does not suffer from the afore-mentioned problems. It does not require requests to go through a particular replica. Furthermore, ID-Replication allows different replication degrees for different key ranges. This allows for using higher number of replicas for hotspots and critical data. We provide detailed evaluation of ID-Replication, and compare it with SL-replication. Our results show that ID-Replication is less sensitive to churn than SL-replication, which makes it better suited for building consistent services and for working in asynchronous networks where inaccurate failure detections are a norm.

Future Work: Due to its low sensitivity to churn, a possible step forward would be to build a consistent key-value store using ID-Replication. Each replication group can act as a replicated state machine, where operations are performed in a total order on the replicas. Since replica groups change with dynamism, we propose using a reconfigurable replicated state machine such as SMART [11].

References

1. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-Grid: a self-organizing structured P2P system. SIGMOD Record 32(3), 29–33 (2003)
2. Arad, C., Dowling, J., Haridi, S.: Developing, simulating, and deploying peer-to-peer systems using the Kompics component model. In: COMSWARE 2009 (2009)
3. Ghodsi, A., Alima, L.O., Haridi, S.: Symmetric Replication for Structured Peer-to-Peer Systems. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 74–85. Springer, Heidelberg (2007)
4. Glendenning, L., Beschastnikh, I., Krishnamurthy, A.: Scalable Consistency in Scatter. In: ACM SOSP, pp. 15–28 (2011)
5. Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in dynamic structured P2P systems. In: Proceedings of the 23rd Conference of the IEEE Computer and Communications Societies (2004)
6. Gummadi, K.P., Saroiu, S., Gribble, S.D.: King: estimating latency between arbitrary internet end hosts. In: IMW 2002: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, pp. 5–18. ACM, New York (2002)
7. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: STOC, pp. 654–663. ACM (1997)

8. Karger, D.R., Ruhl, M.: Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279, pp. 131–140. Springer, Heidelberg (2005)
9. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44, 35–40 (2010)
10. Leong, B., Liskov, B., Demaine, E.D.: Epichord: Parallelizing the chord lookup algorithm with reactive routing state management. Computer Communications 29(9), 1243–1259 (2006)
11. Lorch, J.R., Adya, A., Bolosky, W.J., Chaiken, R., Douceur, J.R., Howell, J.: The smart way to migrate replicated stateful services. In: EuroSys (2006)
12. Muthitacharoen, A., Gilbert, S., Morris, R.: Etna: a Fault-tolerant Algorithm for Atomic Mutable DHT Data. Mit technical report, MIT (June 2005)
13. Narendula, R., Miklós, Z., Aberer, K.: Towards access control aware p2p data management systems. In: EDBT/ICDT Workshops, pp. 10–17 (2009)
14. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
15. Singh, A., Gedik, B., Liu, L.: Agyaat: mutual anonymity over structured p2p networks. Internet Research 16(2), 189–212 (2006)
16. Sit, E., Morris, R.: Security Considerations for Peer-to-Peer Distributed Hash Tables. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 261–269. Springer, Heidelberg (2002)
17. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON) 11(1), 17–32 (2003)
18. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. J. Network Syst. Manage. 13(2), 197–217 (2005)