Simple Replication in Structured Peer-to-Peer Networks

Ahmet Yüksektepe
Email: ahmet.yueksektepe@hhu.de
University of Düsseldorf, HHU - 2015 SS - Opportunistic and Peer-to-Peer Networks

***Abstract*—Peer-to-peer networks grow very fast and nowadays represent about 70% of the Internet traffic [1]. They offer great advantages for distributed collaborative applications depending on the application. But one key problem in structured peer-to-peer networks with a distributed hash table (DHT) is the fact that the stored data on such a network can easily get lost when the node responsible for storing the data goes offline. If the necessary measures are not taken, the data might not only be temporarily unavailable but also could be lost permanently. In many application scenarios, like a distributed data storage, keeping data online is very important and one of the expected requirements. To avoid this kind of situation, in this paper we introduce the simple replication protocol which is the idea of storing the same data not only on a single node, but also on multiple nodes among the peer-to-peer network. For the purpose of experimenting, the designed protocol has been implemented in the simulation framework Peerfactsim.KOM and simulated on top of the created peer-to-peer network utilizing the DHT protocol Chord. Under different churn behaviours the effectiveness of this protocol has been evaluated for different application scenarios. Multiple experiments have been conducted with the help of the simulation environment to examine many key metrics like the lost data ratio, the number of nodes to replicate to and the network overhead. After we have evaluated our experiments results we found out that the replication degree 3 is sufficient for high availability rates, while offering at the same time efficient results concerning the network load.**

***Index terms*—** peer-to-peer networks, simple replication, replication degree, churn, Chord, Peerfactsim

I. Introduction

Cloud technologies are a very popular topic in the current IT-World and one of the big advantages of cloud is to be able to store your data not just locally but also in the internet. Nowadays the common idea for this purpose is using client-server based hosting services. But this architecture introduces many problems, among other things availability, security or even censorship. The server is obviously the weak point of this design. The idea of peer-to-peer networks eliminates the single point of failure problem in the client-server architecture. In peer-to-peer overlays each participant, each peer in the network is treated equally (that is the basic idea at least) and any application is realized together between the peers. This network stays functional even if nodes leave and join. Therefore peer-to-peer networks have much more potential, also from the resource point of the view. They are more secure and stable. But one of the weaknesses of peer-to-peer overlays in case of a distributed data storage system as the application on top, is the availability problem of nodes in the network. The churn rate of the nodes and the heterogeneous structure of peer-to-peer overlays with different types and capabilities of peers renders it difficult to guarantee the availability of persisted data in such networks. Therefore replication is a very central topic in this research area.

This paper was motivated to design a simple replication protocol as an approach to address the problematic explained above. With the help of the simulation software Peerfactsim.KOM [2] and Chord [3] as the underlying distributed-hash-table algorithm, we simulated a peer-to-peer network, to find out how much the data loss can be reduced with simple replication, and what the optimal values in peer-to-peer networks are to guarantee high-resource availability, without introducing unnecessary costs.

With these intentions, in Section 2, related work in this research domain has been shortly examined, taking the protocol PAST as an example for an already existing protocol for a distributed file system utilizing replication. Afterwards, in Section 3, it is described in detail how our simple replication protocol has been designed, which requirements were set and which strategies were chosen to realise them. In addition, we also give a short overview to how the protocol has been implemented. In section 4, the simulation environment and the simulation scenario are explained. Metrics are introduced to define experiments, and finally experiment results are presented and so the protocol is evaluated. At the end, in Section 5, the conclusion to this paper and in Section 6 future work about the simple replication protocol is located.

II. Related work

A. PAST

In Pastry [4] routing is handled in a tree-like structure. Nodes have so called leaf-sets, and neighborhood lists numerically close to them, helping them realising lookups by routing to numerically nearest nodes-id's. The leaf-set and the neighbor-list are kept updated.

PAST [5] is a distributed file storage protocol which is based on the DHT Pastry. Files are hashed by their name, or any other unique information, and saved at first on the numerically closest responsible node in Pastry. Afterwards,

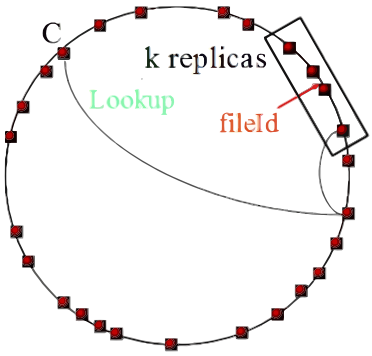

Fig. 1. PAST - Client c find a replica near fileId [6]

the file is replicated to x further nodes numerically around the first hash-id. This means the file is replicated to nodes in the leaf-set and x is limited by the size of the directly reachable nodes. To read a file PAST similarly uses Pastry to find the responsible node for the hash of the file-ID.

The big advantage PAST offers, is that disconnected nodes are detected by Pastry already, so it requires less effort to assign new nodes to replicate to and Pastry itself is the KeepAlive scheme of PAST. But depending completely on the Pastry-overlay and the leaf-sets has also its drawbacks. Therefore a disadvantage of PAST is that all the replications of a file are distributed around a single ID, so the limited key-space on which the file is replicated to, is vulnerable against churn. The nature of the Pastry protocol itself has not a good stabilization against churn and its vulnerable for security reasons (e.g. against routing attacks [7]).

Our first priority was to create a simple and reliable replication protocol, so we have chosen the Chord-Overlay to experiment on, among other things, also due to Chord providing better stabilization against churn.

III. Replication Protocol

A. Overview

Designing a simple replication protocol isnt as simple as the name suggests since there are many parameters and problems one needs to consider. The challenge of accomplishing 100% data availability in a peer-to-peer network is nearly impossible to overcome, the design process of the protocol has started accepting this fact, because simply proven, even if the desired data is fully replicated to all participants of the network and one would assume the most stable situation, in the worst case, all nodes in the network could leave at the same time, never coming back, which would mean the data is lost. So peer-to-peer networks as the underlying system have a very dynamic structure, and therefore the dynamic choice for the amount of nodes to replicate to seemed logical and we created a parameter named Replication-Degree. Already with this variable the replication protocol was greatly able to adapt to different application scenarios (the impact of different Replication-Degrees on the network-load as well as the on the data-availability has been further examined in section 4.3).

Knowing we need to replicate our data to x nodes, another big challenge appears: it is not sufficient to replicate to x nodes once, because the nodes storing the replications(Replication-Nodes) could never come back online once going offline, which is not acceptable, on the grounds we want continuous availability and reliability. So it became clear very fast that replication is rather a continuous process, which has not an end, as long as the data needs to be alive.

At this point we needed to pick up a distribution strategy among the nodes, and a strategy to realise the continuous replication process.

B. Distribution Strategy

There are many ways how the replication of data could be distributed among peer-to-peer nodes. One practice concerning this problem is for example using Multiple Hash-Functions for each replication, which means creating multiple DHTs ([8]). Another idea is using distribution functions regarding locality information, or even using ID numerically close to the calculated first DHT-Key like in PAST.

Our distribution strategy, and at the same time our storage strategy for the redundant data we wanted to create was to use the hash-function of the DHT, on the unique identifier of the data multiple times (Sequential hashing, also Chain hashing), so that ID_0=hash(data-UID) and ID_x=hash(ID_(x-1)). This means that the replications share the same set of hash-representations. In the following example it is illustrated how the Replication-Nodes are determined in our protocol, in case of for example Replication-Degree=3:

```
Node 0=lookUp(Sha1(Data.UID))
Node 1=lookUp(Sha1(Sha1(Data.UID)))
Node 2=lookUp(Sha1(Sha1(Sha1(Data.UID))))
```

As it can be seen the unique data-ID is being hashed multiple times with the same hash-function Sha1. The lookUp operation gives us the closest node-ID responsible in the DHT for the given data-ID.The data is then replicated to each responsible node.

In contrast to the alternatives this strategy is pretty simple and has some advantages. With the sequential hashing strategy, thanks to the nature of the hash-function of the DHT, nothing new other than the existing DHT is required. Basically, this way each hash generates completely different values and a more stabilized replication network can be expected, for the reason that the replications are distributed in a heterogeneous manner among the participants of the DHT network.

C. Replication Strategy

To have continuously a set of online Replication-Nodes, we needed kind of a ping, a KeepAlive mechanism between the nodes. So the first idea we had concerning the replication strategy was to choose a MasterNode among the calculated responsible nodes, which has the task of continuously maintaining the replications on all Replication-Nodes of the corresponding Data. This MasterNode would then check in certain time intervals if every Replication-Node has the replication, or if every Replication-Node is online with the help of short messages. As soon as an offline node has been detected, a new responsible node would be contacted. This way the number of online Replication-Nodes would be held constant. Unfortunately in many scenarios this strategy is very unstable. In case the MasterNode goes offline, the new MasterNode has eventually not the replication of the data yet, and the other nodes in the Replication-Node-Set are not obligated to inform the MasterNode that the data even exists. As it can be seen, the idea of a single node responsible for others is against the nature of a peer-to-peer network, because in such architectures participants of the network usually have equal rights. As an example the Kademlia [9] protocol uses this replication strategy. For the purpose of File Sharing in peer-to-peer networks this scheme can be sufficient because the data loss can be tolerated and in File-Sharing protocols like BitTorrent [10] rather centralised nodes like the Tracker-nodes fulfill this task, which are assumed to be more robust. But in case of a distributed data storage the availability possible with a single MasterNode only is not acceptable

On the other hand, a much more robust solution to the problems presented above, is to declare each Replication-Node as a MasterNode at the same time. This way, as long as one Replication-Node is alive, it serves as a distributor and replicates the data to all other Replication-Nodes. As a chain-reaction, after receiving the replication, each Replication-Node also tries to inform all others. This Multi-MasterNode strategy has far more network-overhead, but it is indeed able to guarantee the convergence of the replication and the availability of the data in the system which is the reason why we did prefer this replication strategy over the Single-MasterNode Strategy.

Figure 2 explains how the replication is realised, with each Replication-Node responsible for Data X flooding all other Replication-Nodes with an Update-Message, to which they need to reply to inform the asking node about being alive and having the replication already or not knowing anything about the replication yet. In case the replication is not present, through a Store-Message the replication is sent by any Replication-Node.

D. Create, Read, Update and Delete

With the protocol design we introduced above, we wanted to achieve the operation set: Create, Read, Update and Delete

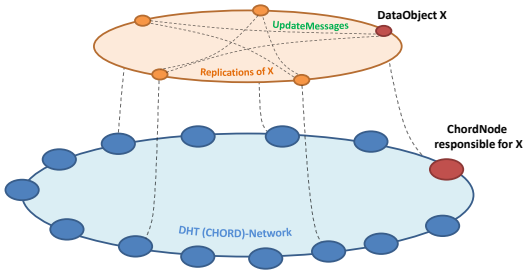

Fig. 2. Simple Replication - UpdateMessage Flooding - Replicationgrade: 5

(CRUD) among the Replication-Nodes. Especially the Update and the Delete operations introduced further challenges at this point. In case a modification to the replicated data was possible and the data was changed by two of the Replication-Nodes concurrently, consistency problems would occur, and a Read operation would result in different data for the same data-ID from different-Replication-Nodes.

To address this problem, we needed to be able to differentiate between two updates of a data with one unique-ID and for this reason we needed to store additional data on Replication-Nodes. With introducing the metadata version we were able incrementally update the data. While we could have chosen to give a completely new ID for a modified data and see it as a completely new data, the Version-Control method was far more useful for keeping the context of the data and therefore more pragmatic from the application layer point of view. In light of the given condition, it is described in the following how the CRUD operations are realised in our simple replication protocol:

CREATE
The create operation can be seen as a special case of the update operation. If the data itself, and the meta data like the version are not existing yet, the first responsible node in the P2P-Network found with the corresponding DHT-ID stores them and gets to be the first Replication-Node of this data. As soon as the initial storage is done, an Update-Operation is started for the data, with the version equal to 0.

READ
The node possessing the data-ID just calculates the first responsible Replication-Node and requests the data. The Replication-Node responsible for the first Hash in the sequential hash chain, answers to the request with the data and if necessary (in case this node has not the fully replicated state of the data yet) gives the direct address of another Replication-Node. In case the first responsible Node cant be reached, the next Node in the hash chain is being contacted, until one Replication-Node is found.

UPDATE
The Update-Operation is the core functionality of the protocol. If the data has been modified, or a modified version of the data has been received, the Replication-Nodes start the Update-Operation. The Replication-Node at which the updated data arrives calculates depending on Replication-Degree the other uninformed Replication-Nodes and sends an Update-Message to them, initialising the communication. Comparing the version of the data, the Replication-Nodes are able to avoid inconsistency and decide if the incoming update is legit. After a Replication-Node answers with an UpdateResponse stating they don't have this [version] of the data yet, a Store-Operation is started, and the data is fully replicated, and in case the contacted Replication-Node has a newer version, a request message is sent to this Replication-Node immediately for being updated. As soon as any ReplicationNode gets updated, to accelerate the convergence of the replication update, the updated ReplicationNode starts an Update-Operation for each ReplicationNode other than itself and the Updater-Node.

DELETE
Although at first sight deleting a data seems to be the most trivial thing, the delete operation is quite challenging to realise, because it is not sufficient to just delete all online replications, as Replication-Nodes can turn back online, and don't posses the information of the data being deleted, and furthermore these nodes would even continue to replicate the deleted data. To be able to guarantee the deletion process, we defined a special case of the update process and used the Update-Operation to realise deleting. So the Delete-Operation is realised in the following way in our protocol:

In case the version number of the data is set to its maximum value, the data is declared as deleted. So, if the max-version is detected, the actual data is deleted and only the metadata to the data is kept. The metadata is continued to be replicated as the whole deletion is seen as a regular update increasing the version number. This strategy achieves to converge the deletion. After a certain time period the data could be deleted including the metadata. But the longer the deletion information is kept, the less is the risk of the data re-appearing.

E. Implementation Overview

The illustration 3 presents the resulting implementation of our simple replication protocol. For realising the replication protocol the Replication-Adapter has been developed, as the core of the implementation. Replication-Adapter is a module which can be attached to each Chord-Node. It scans the DHT entries of the node for instances of Replication-MetaData. For each found Replication-MetaData the Replication-Adapter starts the replication process. Finding first the responsible DHT-Nodes with the help of the hash-function and the lookUp-Method of the DHT, the Adapter starts for each responsible node an Update-Operation. The process of updating

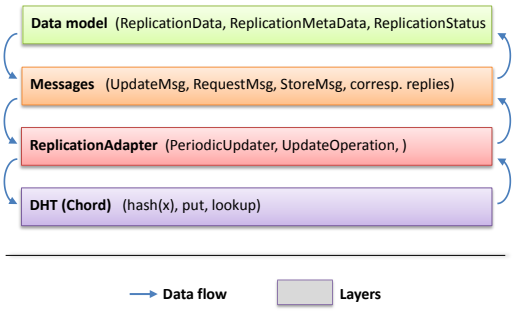

Fig. 3. Simple Replication - Protocol Layers

repeats continuously in a predefined time-interval with the help of the Periodic-Update-Operation which is our Multi-MasterNode KeepAlive mechanism as well as the update-initiator. The Replication-Adapter keeps track of the addresses of Replication-Nodes to not each time look them up, and automatically resolves node failures with the help of the Update-Operations. The communication is realised through request- and reply-messages which indicate if replication is necessary or not corresponding to the given Replication-Status.

IV. Evaluation

A. Simulation

To evaluate the protocol we created different experiments which consisted of representant application scenarios with different metrics. After finding the right metrics settings for each scenario, we simulated each experiment with the help of our implementation in the Peerfactsim.KOM framework. For conducting our experiments we first created a special environment with the help of the configuration-file of our simulation in Peerfactsim.KOM and each experiment equaled a special version of the simulation with different key metrics.

The simulation is the result of many important network settings and configurations in the Peerfactsim.KOM framework. We tried to simulate an environment as similar to the internet as possible. So we have built in many real world behaviour like Jitter, Packet loss, IP-Fragmenting, Latency etc. to make the simulation as realistic as possible.

Furthermore we defined hosts joining the simulation at different stages of the simulation. Each host is able to communicate with other hosts due to the their application stack consisting of the NetLayer (as IPv4), Transport-Layer (as TCP/UDP), Overlay-Layer (as Chord with our Replication-Adapter-Attachment which is representing our simple replication protocol) and an Application-Layer (our DataGenerator). So, by joining the network, each host generates new Data with the DataGenerator and commits it to the Chord network. This way we were able to control how much data is brought in by each host to the simulation.

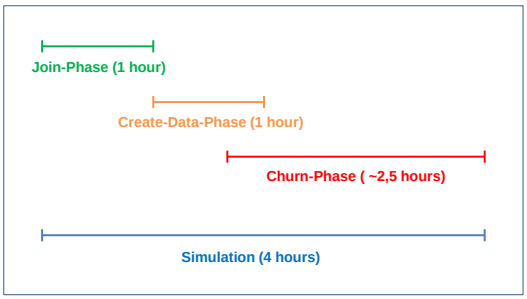

Fig. 4. Simulation Phases

In addition to these basic components, we added a Churn-Generator to the simulation, which gives true meaning to our experiments, as we wanted to observe how the simple replication protocol reacts to the presence of churn and how far the availability of data is impacted hereby. We used two different variants of the Churn-Generator, the Exponential-Distribution-Model and the Kademlia-Model (KAD-Model which is a DHT based on Kademlia). In case of the Exponential-Distribution-Model the churn distributes exponentially with time and we were able to configure the churn behaviour of the this model. The Kademlia-Model on the other hand was based on measurements of a real Kademlia usage in a P2P-File-Sharing application [11].

And at the end, we defined a scenario for the simulation. The events and actions in the simulation scenario are shown in figure 4. The simulation has following Phases:

1. Join-Phase:
nodes join the network for about 1 hour

2. Data-Creation-Phase:
in the interval of 1 hour each node creates the desired amount of data

3. Churn-Phase:
in the middle of the Data-Creation-Phase the Churn-Generator starts creating churn until the end of the simulation, the Churn-Phase takes in average about 2.5 hours

Taking this course of actions, a realistic situation in a peer-to-peer network has been simulated because as soon as the churn phase begins, there is already existing replication data, and some new data is joining, while the churn behavior is present.

B. Experiment Setups

Moreover, for the simulation we needed to define constants in our protocol. Depending on default Chord-Implementation timeout values and regarding the estimated average session time of a peer-to-peer node, we defined the time interval between two Periodic-Update-Operations as 5 minutes, for nodes to be able to still update before disconnecting. In addition, we defined the data size as 15 KB. On the other hand we left following metrics variable in order to create different experiment setups and to evaluate how the protocol performs under different conditions:

-Node-Count
-Data-Count
-Replication-Degree

Moreover, concerning the exponential-churn-distribution model, we had these variables:

-Churn-Factor (which is the amount of nodes being unavailable)
-Churn-MeanSessionLength in minutes

For simulating different application scenarios, we created 14 different experiments consisting of 3 different experiment groups:

1. Experiment Group (experiments 1-6)
In the first group of experiments we did set the Replication-Degree to 3 and we started the first experiment with 10 nodes and increased this number up to 1000 nodes, while also increasing the data count. For 6 experiments, we switched randomly between the exponential churn distribution model and the kademlia model.

We considered the experiments with higher amount of nodes rather as publicly accessible distributed peer-to-peer applications (for example a P2P-Wiki) while on the other hand we regarded experiments with lower node-counts as rather private peer-to-peer applications in more controlled environments, like a distributed file system in a company. On this grounds, in case of the exponential churn distribution model and low data count, we have set the Churn-Factor rather low, and the Churn-MeanSessionLength rather high, while in experiments with higher node counts we took higher Churn-Factor and less Churn-MeanSessionLength because each participant had a higher risk of leaving having less incentives for being a part of the application.

2. Experiment Group (experiments 7-12)
For the second experiment group we started with Replication-Degree 1 and increased it up to 6, to measure which Replication-Degree is sufficient for a good availability ratio.

3. Experiment Group (experiments 13-14)
After optimizing our implementation, in this experiment group we measured how the protocol performs and scales for some larger values like replication degree=8 and data count=4000.

	Experiment	1	2	3	4	5	6	7	8	9	10	11	12	13	14
parameters	Node Count	10	25	50	250	500	1000	1000	1000	1000	1000	1000	1000	1000	1000
	Data Count	100	250	500	1000	1500	2000	2000	2000	2000	2000	2000	2000	4000	4000
	Replication Degree	3	3	3	3	3	3	1	2	3	4	5	6	7	8
	Churn Factor	0.2	KAD	0.3	0.4	KAD	0.5	KAD	KAD	KAD	KAD	KAD	KAD	KAD	KAD
	MeanSessionLength in min.	30	KAD	20	15	KAD	15	KAD	KAD	KAD	KAD	KAD	KAD	KAD	KAD
results	**Data Availability**														
	Online node count	9	13	36	146	311	516	610	569	595	579	612	579	583	581
	Available data count	100	250	500	999	1500	1996	1265	1983	1998	2000	2000	2000	4000	4000
	Available data in %	100	100	100	99.90	100	99.80	63.25	99.15	99.90	100	100	100	100	100
	Unavailable data in %	0	0	0	0.10	0	0.20	36.75	0.85	0.10	0	0	0	0	0
	Network Costs														
	storeOperation Count	623	1104	9402	10073	7676	18548	1948	6428	9539	14025	17477	20909	92183	160767
	storeOperation Size in MB	9	17	72	155	118	286	30	99	147	216	269	322	7107	12396
	updateOperation Size in MB	14	41	103	221	298	426	2	134	399	802	1337	1965	6323	8447
	lookUpOperation Size in MB	4	6	39	81	52	143	3	26	64	135	215	309	2493	3783
	totalMessage Size in MB	28	65	215	458	469	856	35	260	611	1155	1821	2596	15924	24627
	totalMessage Count per min.	271	664	2141	3439	4811	8405	129	2275	6304	12728	20902	30505	13478	192057
	totalMessage Size per min.	0.19	0.43	1.43	3.05	3.13	5.71	0.23	1.73	4.07	7.70	12.15	17.31	106.16	164.18

Fig. 5. Simple Replication - Experiment results

C. Experiment Results

Under these circumstances, the table 5 emerged, which contains the experiments (including the parameter decisions) and the resulting output-metrics. As it can be seen we measured in fact two essential metrics: how much additional network traffic has been caused through simple replication and how much data was available with each given Replication-Degree.

The experiment results indicate that the Replication-Degree 3 is able to offer very high availability. But especially in very dynamic scenarios with greater amount of nodes, and at 40-50% churn rates, there is still minor data loss. At the end, replication degree 4 was sufficient for 100% data availability in our experiments. At the same time, as it can be seen in figure 6, the results confirmed correctly that the network costs are increasing exponentially with an increasing Replication-Degree n.

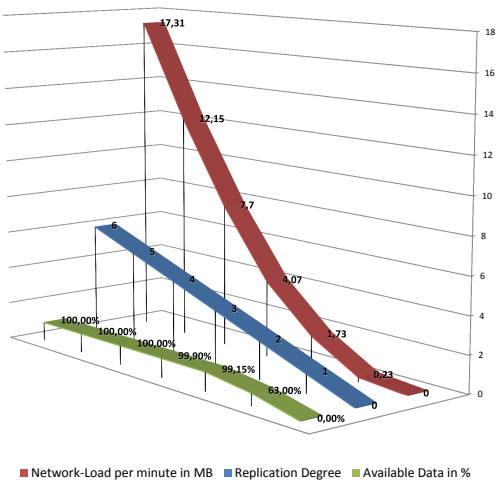

Fig. 6. Replication-Degree - Effects on expermient outputs

During the simulation the protocol created for example in experiment 9, 4.07 MB per minute network-load, with the data size constantly set to 15 KB and a store-Operation count of about 10000. This value seems acceptable but with greater data sizes like for example 1 MB each data, the network-load per minute increases up to 1 GB per minute. So the configuration of the protocol should be very application specific to stay scalable.

On the other hand, taking into consideration that the experiment results are heavily affected by the initial replication costs, we could claim that the maintaining costs are relatively low because with each disconnect, and a reassignment of a replication a new copy of the data is being cached by the old responsible node.

Furthermore, to screen the difference between the churn models we utilized for our experiment and the impact the protocol had on the network-load in combination with each model, figures 7 and 8 show the throughput of Bandwidth in the network during the simulations with each model. On these two figures we can see very well how the protocol tries

to recover with nodes leaving the network. As soon as churn begins the network traffic peaks. Whats more, we also see that the protocol is able to recover pretty fast in both churn models while it performs better with the kademlia model, because the exponential model is configured here to have much higher churn rates and less session-time for the nodes in the network which explains the higher amplitudes in the network-load when utilizing the exponential model.

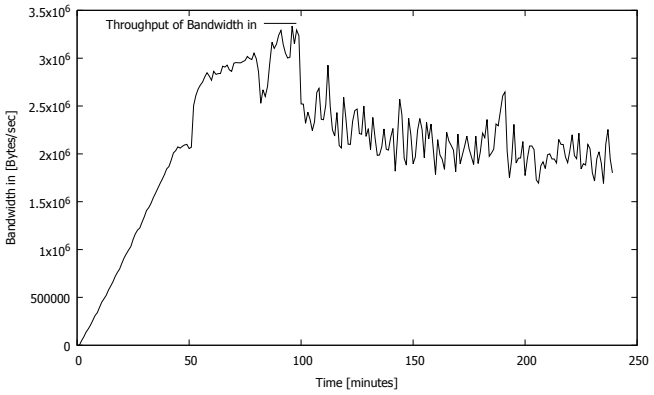

Fig. 7. Experiment 6 - network-load - exponential churn

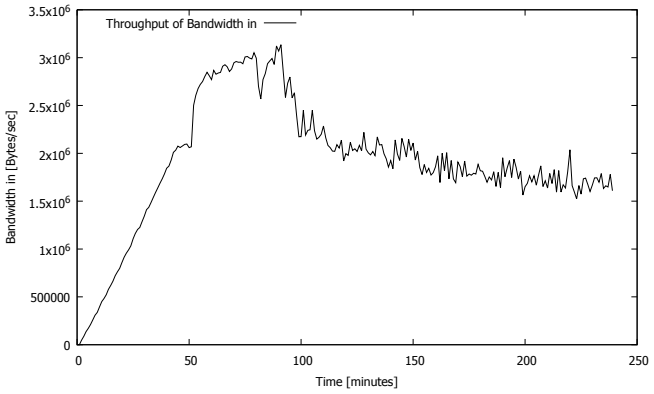

Fig. 8. Experiment 9 - network-load - Kademlia churn

Apart from the results above, an additional metric we observed at the end of the simulation in each experiment was the amount of data which were fully replicated to the maximum replication degree. We noticed that in all experiments, the amount of maximally replicated data was at best about 90%. This outcome confirms that the continuous churn behaviour in the network results in a continuous replication process and the replication is rarely converged to 100% any time during the simulation. But especially in the third experiment group, with replication degree 7 and 8, we measured that the amount of fully replicated data sunk drastically, up 60%. This simply indicates that there is a maximum number of rational replication degree given a certain churn rate and a maximum number of nodes, because the protocol won't have sufficient time to ever converge for greater replication degree values, as the replication process itself is limited by time and network speed.

V. Conclusion

In case of a distributed collaborative application in a structured peer-to-peer network, it is a big challenge keeping data available due to the highly dynamic nature of peer-to-peer networks. To overcome this challenge and to ensure availability of data permanently, we designed a simple replication protocol, creating multiple copies of data among the participants of the peer-to-peer network and assigning them also the task of replicating the data further, up to a predefined Replication-Degree. We called this continuous process the Multi-MasterNode replication strategy. Moreover, we distributed the replications with sequential hashing. We conducted experiments based on our simulation in the software environment Peerfactsim.KOM on top of the Chord-DHT to find an optimal value for the Replication-Degree setting, and to measure which network costs arise subject to simple replication.

The experiment results have shown us that while the distribution strategy we chose had the big advantage that the nodes were chosen heterogeneously, accomplishing a relatively stable distribution, the disadvantage of sequential hashing was requiring a new lookup for each ReplicationNode, as this scheme does not use the built-in DHT Infrastructure to gain information about the whereabouts of the responsible nodes. This disadvantage leads to an increase in the amount of lookUp operations, which simply creates more network traffic overhead. Besides increasing the reliability incredibly, the Multi-MasterNode strategy also generates additional traffic. The n-to-no scheme results in flooding of the network with Update-Messages.

Although the traffic caused by the necessary flooding of the network limits the application domain of the simple replication protocol and the protocol has shortcomings for sure due to the nature of replication, we had surprisingly successful and useful results for applications where the availability of data, and therefore the replication, is critical. With the help of various experiments we found out, that already the Replication-Degree 3 offers very high reliability while Replication-Degree 4 almost guarantees availability. Yet, these values need adaption, as they can be too much or even too unreliable depending on usage scenario.

VI. Future Work

Taking a further step, the simple replication protocol we have developed could evolve into a more complex protocol. During our experiments we had the idea that it would be very fitting, if the protocol itself could observe the churn behaviour and adapt its Replication-Degree or its replication strategy according to the situation. Introducing this kind of self-awareness and similar control mechanisms could improve the efficiency of the protocol a lot. Another idea we had, is

to be able to set different Replication-Degrees for each data. This kind of configurability would also allow for a broader range of possibilities. Moreover, we are interested in how the protocol performs on top of other DHTs than Chord. It is surely valuable to know, which DHT protocol fits the simple replication protocol best.

At the end, we believe, although the simple replication protocol is very useful and the implementation was efficient due to our test results, it has not limitless application domain. With data sizes greater than only 15 KB, the network costs caused by the replication protocol can raise rapidly. Therefore extending the capabilities of simple replication with functionality for shifting the burden of replication from network load (appearing by fully copying the data) rather in the direction of the computing power can be more intelligent regarding dynamic applications with high churn rates.

REFERENCES

[1] M. Y. Talha, R. W. Aldhaheri, M. H. Awedh *et al.*, "Performance study of locality and its impact on peer-to-peer systems," *Communications and Network*, vol. 7, no. 01, p. 1, 2015.

[2] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator Experiences and Lessons Learned," in *IEEE P2P '11: Proc. of the Int. Conf. on Peer-to-Peer Computing*, 2011.

[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *SIGCOMM '01: Proc. of the Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001.

[4] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM Middleware '01: Proc. of the Int. Conf. on Distributed Systems Platforms*, ser. LNCS, vol. 2218. Springer, 2001.

[5] P. Druschel and A. I. T. Rowstron, "PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility," in *IEEE HotOS '01: Proc. of the Workshop on Hot Topics in Operating Systems*, 2001.

[6] Nathaniel Crowell, "PAST: P2P Storage Utility," 2007. [Online]. Available: https://www.cs.umd.edu/class/spring2011/cmsc818k/Lectures/Pastry-Past.pdf

[7] F. A. Eichert, M. Monhof, and K. Graffi, "The impact of routing attacks on pastry-based p2p online social networks," in *Euro-Par 2014: Parallel Processing Workshops*. Springer, 2014, pp. 347–358.

[8] R. Kapelko, *Towards fault-tolerant chord P2P system: Analysis of some replication strategies.* Berlin: Springer, 2013, pp. 686–696.

[9] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *IPTPS '02: Proc. of the Int. Workshop on Peer-To-Peer Systems*, ser. LNCS, vol. 2429. Springer, 2002.

[10] B. Cohen, "BitTorrent," http://www.bittorrent.com.

[11] M. Steiner, T. En-Najjary, and E. W. Biersack, "Analyzing peer behavior in kad," *Institut Eurecom, France, Tech*, 2007.