Analytical Study on Improving DHT Lookup Performance under Churn

Di Wu, Ye Tian and Kam-Wing Ng
Department of Computer Science & Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{dwu, ytian, kwng}@cse.cuhk.edu.hk

Abstract

The phenomenon of churn degrades the lookup performance of DHT-based P2P systems greatly. To date, a number of approaches have been proposed to handle it from both the system side and the client side. However, there lacks theoretical analysis to direct how to make design choices under different churn levels and how to configure their parameters optimally. In this paper, we analytically study three important aspects on improving DHT lookup performance under churn, i.e., lookup strategy, lookup parallelism and lookup key replication. Our objective is to build a theoretical basis for DHT designers to make better design choices in the future. We first compare the performance of two representative lookup strategies - recursive routing and iterative routing, and explore the existence of better alternatives. Then we show the effectiveness of parallel lookup in systems with different churn levels and how to select the optimal degree of parallelism. Due to the importance of key replication on lookup performance, we also analyze the reliability of replicated keys under two different replication policies, and discuss how to make configuration in different environments. Besides analytical study, our results are also validated by simulation, and Kad [1] is taken as a case to show the meaningfulness of our analysis.

1 Introduction

In recent years, a number of peer-to-peer (P2P) systems (e.g., Naspter, Gnutella, KaZaA, BitTorrent, Kad, etc) have been invented and widely deployed. The P2P model now emerges as a dominant paradigm over the Internet, and greatly changes the communication style between users. Among various P2P systems, DHT-based (Distributed Hash Table) design is taken as a promising approach to build a simple and yet efficient infrastructure for P2P applications.

For real deployment, however, system designers have to tackle many new challenges. As we know, a P2P system is a decentralized and dynamic system, in which peers continue to join and leave. This phenomenon of peer dynamics is called "*churn*", and may cause lookup failure or data key loss. To improve the DHT performance under churn, researchers have proposed various kinds of approaches (e.g., [2], [10], [5], [6], etc) in the aspects of lookup strategy, neighbor selection, data redundancy, parallel lookup, failure recovery, etc. However, the effectiveness of these approaches is mostly verified via simulation or empirical study, and there is few analytical study on these design choices and their related parameter settings. This motivates us to perform analysis to get a deeper insight of the proposed design solutions, explore their optimal parameter configuration and investigate the existence of better alternative approaches.

In this paper, we perform an analytical study on three important aspects on improving DHT lookup performance under churn:

- *Lookup Strategy*: We compare the performance of two representative lookup strategies - *Recursive Routing (RR)* and *Iterative Routing (IR)*, and deduce the closed-form expressions of their performance metrics. With the theoretical results, we can know exactly which strategy is better and how good it is for a given setting. It is observed that, both *Recursive Routing (RR)* and *Iterative Routing (IR)* can only perform well under a certain range of churn levels. To make the lookup strategy perform the best under the whole range of churn levels, two enhanced alternatives are considered. We find that, a simple *Two-Phase Routing Strategy (RR+IR)* can well exploit the benefits of both recursive routing and iterative routing. Besides, we also propose another new lookup strategy, in which recursive routing is enhanced with an ACK mechanism. It further improves the lookup performance, and behaves better than *RR, IR* and *RR+IR*.
- *Lookup Parallelism*: We provide the closed-form expressions for the performance metrics of parallel

Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)
0-7695-2679-9/06 $20.00 © 2006 IEEE

lookup, and show the effectiveness of parallel lookup under different churn levels. Considering the tradeoff between the lookup latency and the message overhead, our theorem can help the designers determine an appropriate parallelism degree.

- *Lookup Key Replication*: We study two typical policies of key replication to improve lookup performance under churn: (i) replication without repair mechanism; and (ii) replication with repair mechanism. We formally deduce the expected lifetime of the replicated key under different policies. It is found that, whether a repair mechanism is necessary depends largely on the peer lifetime distribution, instead of the system churn level. Under exponential lifetime distribution, a repair mechanism is required no matter the churn level is high or low; but under Pareto lifetime distribution, if there exists a very heavy tail, simple replication is enough to guarantee a rather long lifetime of the replicated key. It is true even under a high-churn environment. As to the selection of the replication factor and the repair rate, it is application-specific and we discuss some practical considerations (e.g., data type, data lifetime, etc) during selecting the proper parameters.

Our theorems make it possible for DHT designers to know how to make the best design choices under a given churn level precisely, instead of based on intuitions. Besides analytical study, we also perform simulation to validate our results. Due to the space limit, please refer to [14] for the simulation details and the proofs of our theorems.

The remainder of this paper is structured as follows. We will first introduce the related work in Sec. 2. The model and analysis are presented in Sec. 3. Finally, Sec. 4 summarizes the whole paper.

2 Related Work

Churn makes P2P systems different from traditional distributed systems, and brings new challenges to system designers. To better understand their characteristics and impact, quite a few research works have been conducted in the recent years. Sariou [11] and Daniel et al. [12] performed detailed measurements about real P2P file-sharing systems, including Napster, Gnutella, BitTorrent, Kad, etc. They found similar phenomena of system churn across these systems, and the peer lifetime follows a heavy-tailed distribution.

To mitigate the effects of churn, a number of solutions have been proposed. In [10], Sean et al. proposed to handle churn by reactive or periodic failure recovery, accurate calculation of lookup timeout and proximity neighbor selection. Later, Simon et al. [5] further introduced the concept of K-consistency and designed a failure recovery protocol for K-consistent P2P networks. In [2], Frank et al. explored a range of solutions, such as iterative or recursive routing, proximity neighbor selection, replication, erasure coding, etc, in their implementation of the DHash system. In [7], Li et al. provided a PVC (Performance vs. Cost) model to evaluate different DHT design tradeoffs under churn by their self-developed simulator - p2pSim. Besides performing improvement on existing DHTs, some new DHTs are designed particularly for churn-intensive environments, such as Accordian [6], etc. Different from the above simulation-based study, Daniel et al. [13] developed measurement tools to study the impacts of churn in the real Kad system [1].

There also exist some analytical studies about DHT performance under churn. However, most of them mainly focus on one special kind of DHT. In [8], David et al. provided asymptotic performance bounds of Chord under churn. In [3], [4], El-Ansary, Krishnamurthy et al. analyzed the performance of Chord by using a master-equation-based approach. Our analysis differs in that we consider the common issues in DHT designs, and our objective is not just to analyze the impact of churn, but to help system designers make wise decisions on better handling churn.

3 Modeling and Analysis

3.1 System Model

To make our analysis tractable, some necessary assumptions are made as follows: first, we assume that the underlying links are ideal, and there is no packet loss during routing. Therefore, the routing failure is only caused by stale contacts in the routing table due to system churn; second, to model the system churn, each arriving peer is assumed to be associated with a lifetime L, which is a random variable following certain distributions.

When a peer v joins the system, it will select (or be assigned) a set of existing nodes as its neighbors. Here, the peer selection is assumed to be independent of the peer lifetime. Define R to be the residual lifetime of one neighbor since the peer v joined. According to [9], the CDF (Cumulative Distribution Function) of the residual lifetime R is given by:

$$F_R(x) = P(R < x) = \frac{1}{E[L]} \int_0^x (1 - F(z))dz. \quad (1)$$

where $F(x)$ is the CDF of peer lifetime L, and $E[L]$ is the expectation of peer lifetime L.

A neighbor leaves the system when its residual lifetime is exhausted. In that case, it will take some time S for the peer v to detect the failure and replace it with a live peer. During this period, that contact in the routing table of peer v is in the "*stale*" state; correspondingly, other contacts that

are not in the "*stale*" state are referred to as "*fresh*" contacts. If peer v forwards a lookup to a stale contact, the lookup will fail and cause a timeout. Under the steady state, the probability that a contact is *fresh* is given by:

$$p = \frac{E[R]}{E[R] + E[S]}. \quad (2)$$

where $E[R]$ is the expected residual lifetime of a neighbor ($E[R] = \int_0^\infty (1 - F_R(x))dx$) and $E[S]$ stands for the expected time for neighbor replacement (i.e., repair time) in case of failure.

In the following section, we will perform analysis on lookup strategy, lookup parallelism and lookup key replication based on the above system model.

3.2 Lookup Strategy

The lookup strategies of DHT systems can be generally categorized into two types: *Recursive Routing (RR)* and *Iterative Routing (IR)*.

Most DHTs (e.g., Tapestry, Pastry, etc) adopt recursive routing in their design, in which intermediate nodes on the routing path will directly forward the query to the node in the next hop, without making any acknowledgement to the originator. Thus, the query message can be routed as quickly as possible, but the originator has no control over the lookup process. If the lookup returns no response, the originator has to wait until its timeout, as there is no knowledge about the real cause.

Contrary to recursive routing, iterative routing enables the originator to control the whole lookup process. In each step, instead of letting the intermediate node forward the query on its behalf, the originator will ask the intermediate node to return the address of the next hop. With the returned address, the originator initiates the query to each successive node on its own, until the destination is reached. Due to its manageable behavior, Kademlia and its variant Kad use iterative routing in their design.

For the reason of generalization, we use a variable l to denote the length of routing path between the source and the destination. The value of l is different in various DHTs, e.g., $O(logN)$ in Chord, $O(1)$ in Onehop Overlay, etc. We denote the average one-hop routing latency by Δ, and the Round-Trip Time by $RTT = 2\Delta$.

In both recursive routing and iterative routing, when the lookup responds with no reply, a new lookup will be relaunched after a timeout. However, they differ in their timeout settings. In recursive routing, the timeout is set as T_l, which should be no less than the time to complete the entire lookup, i.e., $T_l \geq (l+1)\Delta$; in iterative routing, the timeout T_h only needs to be no less than RTT, i.e., $T_h \geq RTT = 2\Delta$.

To measure the lookup performance, two metrics are adopted: (1) the lookup latency W, which is the latency to complete the entire lookup; and (2) the message overhead C, which refers to the number of messages incurred during the lookup process. By applying probability theory, we have the following results [1]:

Theorem 1 *For Recursive Routing (RR), the expected lookup latency is given by*

$$E[W_{RR}] = (l+1)\Delta + \frac{1-p^l}{p^l}T_l$$

and the expected message overhead is given by

$$E[C_{RR}] = (l+1) + \frac{(1-p^l) \times [1-(l+1)p^l + lp^{l+1}]}{p^l(1-p)^2}$$

where l is the length of routing path, T_l is the entire lookup timeout, and p is given in Eqn (2).

Theorem 2 *For Iterative Routing (IR), the expected lookup latency is given by*

$$E[W_{IR}] = 2l\Delta + \frac{1-p}{p}lT_h$$

and the expected message overhead is given by

$$E[C_{IR}] = (\frac{1}{p} + 1)l$$

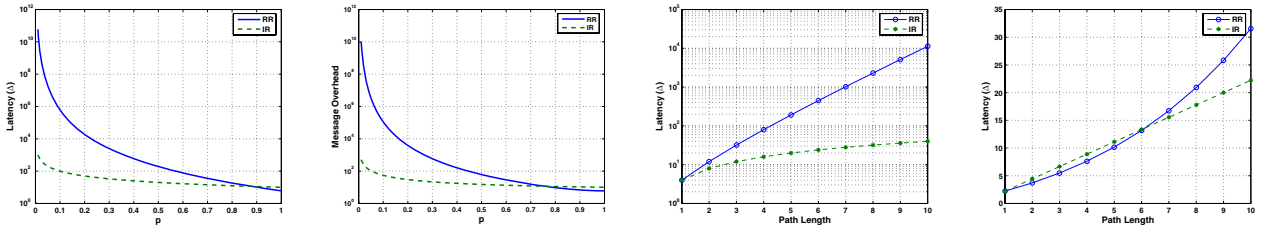
where l is the length of routing path, T_h is the one-hop lookup timeout, and p is given in Eqn (2).

For a better understanding of Theorems 1 and 2, we plot the expected latency and overhead of recursive routing and iterative routing under different p in Fig. 1. Given the expected neighbor replacement time $E[S]$ be fixed, p reflects the level of system churn. When p is large, the churn rate is low; otherwise, when p is small, the churn rate is high. For the timeout setting, we define $T_l = (l+1)\Delta$ and $T_h = 2\Delta$, which are the minimum in the value range.

From Fig. 1(a) and Fig. 1(b), it is found that neither recursive routing nor iterative routing can outperform the other in the whole range of churn levels. When the churn level is low (e.g., $p > 0.9$), recursive routing performs better than iterative routing, and has low latency and low overhead; but its performance deteriorates greatly when the churn level is high. On the contrary, iterative routing is rather robust to system churn, and performs better than recursive routing under intensive churn.

In the above study, the length of routing path l is assumed to be fixed ($l = 5$). However, based on the theorems,

[1] The proofs of all the theorems in this paper are presented in our technical report [14] due to the space limit.



(a) Expected lookup latency (b) Expected message overhead (c) Impact of path length ($p = 0.5$) (d) Impact of path length ($p = 0.9$)

Figure 1: Comparison between *RR* and *IR* (a) Expected lookup latency under different churn level ($0 \leq p \leq 1$); (b) Expected message overhead under different churn level ($0 \leq p \leq 1$); (c) Impact of path length to lookup latency under high churn ($p = 0.5$); (d) Impact of path length to lookup latency under low churn ($p = 0.9$).

l will also impact our comparison results. Fig. 1(c) and Fig. 1(d) show how the path length impacts the routing latency in systems with a high churn rate ($p = 0.5$) and a low churn rate ($p = 0.9$) respectively. From Fig. 1(c), we observe that, when the churn rate is high, the latency of recursive routing increases almost exponentially with the increase of path length, while iterative routing just has a smooth increase. The lookup performance of recursive routing is always less efficient than iterative routing no matter what the path length is. Fig. 1(d) shows that, when the churn rate is low, in case that the routing path length is short ($l \leq 6$), recursive routing can perform better than iterative routing; but for a longer routing path, recursive routing becomes worse again.

Although the above results are not surprising, the meaningfulness of our theorems is that, for a given setting (e.g., churn level, routing path length, etc), we can exactly know which lookup strategy is better and how good it is. It is useful when we make design choices for a special P2P environment.

Based on the observations, we can find that both recursive routing and iterative routing can only function well under a certain range of churn levels. Although we can select a proper lookup strategy according to the system churn level, it will be more ideal if we can devise better approaches that exploit the benefits of them two and perform the best under the whole range of churn levels?

Let us first consider a *Two-Phase Routing Strategy* (*RR+IR*): in the first phase, recursive routing is used for fast routing; if the recursive routing fails, then switches to iterative routing. Its performance is formally given in theorem 3.

Theorem 3 *For Two-Phase Routing (RR+IR), the expected lookup latency is given by*

$$E[W_{RR+IR}] = (l+1)\Delta p^l + [T_l + 2l\Delta + \frac{1-p}{p} l T_h](1-p^l)$$

and the expected message overhead is given by

$$E[C_{RR+IR}] = (l+1)p^l + [\frac{lp^{l+1} - (l+1)p^l + 1}{(1-p)^2} + (\frac{1}{p}+1)l](1-p^l)$$

where l is the length of routing path, T_h is the one-hop lookup timeout, and p is given in Eqn (2).

Besides *RR+IR*, we also propose another lookup strategy, which enhances recursive routing with an acknowledgement (*ACK*) mechanism. It is called *Recursive Routing with ACK Strategy (RR+ACK)* .

The basic idea of *RR+ACK* is that, besides normal forwarding like recursive routing, an intermediate node also sends an ACK containing the addresses of the next hop to the originator. In case of failure, the originator can reinitiate another lookup according to the address in the latest ACK. It is not necessary to start the lookup from the very beginning. To realize $RR+ACK$, there should exist redundancy in the routing table. The performance of *RR+ACK* is given by Theorem 4.

Theorem 4 *For Recursive Routing with ACK (RR+ACK), the expected lookup latency is given by*

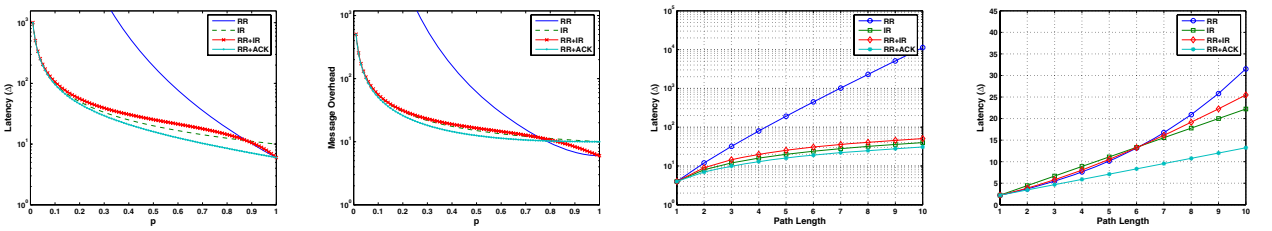
$$E[W_{RR+ACK}] = (l+1)\Delta + \frac{1-p}{p} l T_h$$

and the expected message overhead is given by

$$E[C_{RR+ACK}] = (p + \frac{1}{p})l$$

where l is the length of routing path, T_h is the one-hop lookup timeout, and p is given in Eqn (2).

The performance comparison of *RR, IR, RR+IR* and *RR+ACK* is plotted in Fig. 2. From Fig. 2(a) and Fig. 2(b), we find that, the simple *Two-Phase lookup strategy (RR+IR)* combines *Recursive Routing (RR)* and *Iterative*



(a) Expected lookup latency (b) Expected message overhead (c) Impact of path length ($p = 0.5$) (d) Impact of path length ($p = 0.9$)

Figure 2: Comparison among *RR, IR, RR+IR* and *RR+ACK* (a) Expected lookup latency under different churn level ($0 \leq p \leq 1$); (b) Expected message overhead under different churn level ($0 \leq p \leq 1$); (c) Impact of path length to lookup latency under high churn ($p = 0.5$); (d) Impact of path length to lookup latency under low churn ($p = 0.9$).

Routing (IR) rather well. In low-churn environments (e.g., when $p \geq 0.9$), *RR+IR* has the comparable lookup latency to recursive routing, and better than iterative routing; when the churn becomes intensive (e.g., when $p < 0.9$), it is only slightly worse than iterative routing, but much better than recursive routing. However, the best one is *Recursive Routing with ACK (RR+ACK)* strategy. It has the lowest lookup latency among the four in the whole range of churn levels, and comparable message overhead to iterative routing. But *RR+ACK* will introduce a bit more complexity than *RR+IR* during implementation.

In Fig. 2(c) and Fig. 2(d), we also study the impact of path length on the lookup latency of four lookup strategies under high-churn and low churn environments separately. Under both environments, *IR*, *RR+IR* and *RR+ACK* are robust to the variation of path length, and only *RR* deteriorates greatly. *RR+ACK* has the best performance in all the cases. In fact, *RR+IR* and *RR+ACK* can be seen as the variants of *IR*. Therefore, they have similar robustness to the system churn and the variation of path length.

Besides analytical study, our results are also validated via simulation. The simulation details are available in our technical report [14].

3.3 Lookup Parallelism

Lookup parallelism can be used to improve the lookup performance under churn. In a parallel lookup, the originator initiates multiple lookups simultaneously. Even if some of them meet stale contacts, the whole lookup process can continue to progress without being blocked. Typically, parallel lookup is used together with iterative routing (e.g., Kademlia, etc).

What we want to know is, how much efficiency can we gain from parallel lookup? As the performance improvement of parallel lookup is at the cost of more message overhead, another straightforward question arises that, given a certain level of system churn, what is the optimal degree of parallelism?

In each routing hop except the last hop, we assume k lookups are initiated concurrently and k is referred to as the "*Parallelism Degree*". Every successful lookup will return a list of contacts in the next hop, and here the number of returned contacts by one lookup is assumed to be bigger than the parallelism degree k. After receiving the reply, the originator can initiate another k lookups for the next hop. The process will be repeated until the destination is reached.

Using the same metrics as Sec. 3.2, our results for the performance of parallel lookup are as follows:

Theorem 5 *For parallel Iterative Routing (pIR), the expected lookup latency is given by*

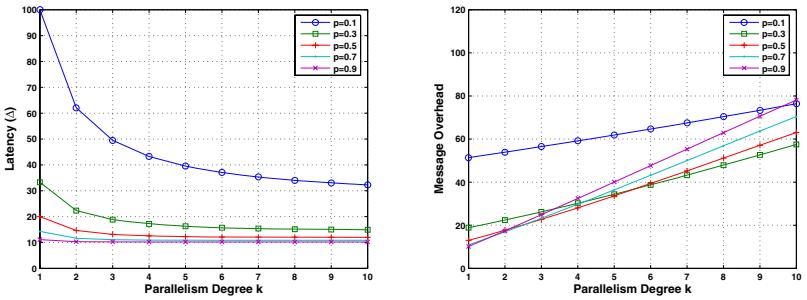
$$E[W_{pIR}] = 2l\Delta + [\frac{(1-p)^k}{1-(1-p)^k}(l-1) + \frac{1-p}{p}]T_h$$

and the expected message overhead is given by

$$E[C_{pIR}] = kp(l-1) + \frac{k(l-1)}{1-(1-p)^k} + \frac{1}{p} + 1$$

where k is the parallelism degree, l is the length of routing path, T_h is the one-hop lookup timeout, and p is given in Eqn (2).

Based on Theorem 5, we plot the expected latency and overhead of parallel lookup under different parallelism degree and system churn level in Fig. 3. It is observed that, parallel lookup can improve the lookup performance for all situations, however, their performance improvements are different. In a high-churn environment (e.g., $p = 0.1$), the use of parallel lookup is very effective; while in a low-churn environment (e.g., $p = 0.9$), its improvement is slight. As to the overhead, Fig. 3(b) shows that, with the increasing of parallelism degree k, the message overhead will increase almost linearly. It is interesting to find that, given $k = 10$, the message overhead is comparable in systems with low



(a) Expected Lookup Latency (b) Expected Message Overhead

Figure 3: Impact of Lookup Parallelism (a) Expected Lookup Latency; (b) Expected Message Overhead.

churn rate ($p = 0.9$) and high churn rate ($p = 0.1$). The reason lies in that we also take the reply messages into account. Under a low churn rate, more reply messages will be generated; while under a high churn rate, the message overhead is mainly caused by resending messages.

From the observation in the above figures, we can clearly see that there exists a tradeoff between the latency $E[W]$ and the overhead $E[C]$. The selection of optimal parallelism depends on the balance between these two factors during design consideration.

Let us take a real P2P system Kad as a case for study, which is the first widely-deployed DHT file-sharing system. According to the measurement results in [12], the peer lifetime distribution in Kad follows a heavy-tailed distribution with the expectation of 2.71 hours, which can be approximated by a Pareto distribution with $\alpha = 2.107, \beta = 3$. The time for neighbor replacement S is about $4 \sim 19$ minutes, and the routing path length l is approximately 3 in Kad due to random bits improvement. Based on the equations in Sec. 3.1 and Theorem 5, we get the numerical results as shown in Table. 1.

Parallelism Degree	$E[W_{pIR}]$ ($\times\Delta$)	$E[C_{pIR}]$
k=1	6.0410	6.0205
k=2	6.0138	9.9798
k=3	6.0136	13.9660
k=4	6.0136	17.9524
k=5	6.0136	21.9388

Table 1: Expected latency and overhead under different parallelism degree

When $k \geq 2$, it can be observed (in Table. 1) that the latency reduction is very slight compared with the incurred overhead. If the message overhead is not a big issue, it is better to set k as 2 or 3. Daniel et al. [13] gives a similar conclusion ($k = 3$) by empirical study of the Kad system.

3.4 Lookup Key Replication

Lookup key replication is another important issue on improving lookup performance under churn. If each data key is only published on exactly one node, it may be problematic under churn. Considering that, if one node leaves the system without migrating its data keys to other nodes, the data keys stored on that node will be lost permanently. Thus, the following lookups towards the keys in that node will fail also.

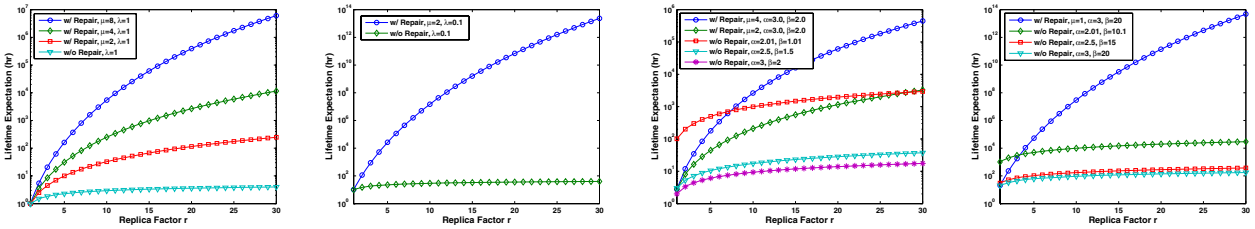
To handle the performance loss due to churn, the most commonly used approach is replication, in which one data key is duplicated in multiple peers. By replication, we can guarantee the lookup to successfully find one replicated key with high probability.

In spite of the fact that replication has already been adopted in practical implementations (e.g., Chord, Kademlia and DHash [2]), some basic questions still require our consideration, e.g., how many replicas are enough to guarantee certain reliability? And do we require repair mechanisms to further improve the robustness to failure? We refer to the above decisions as "*replication policy*".

Two general replication policies are studied in this paper, and they differ in whether a repair mechanism is adopted. In the following, we will analyze them in details.

3.4.1 Replication without repair mechanism

When publishing a data key, the key is replicated at multiple nodes, each of which keeps only one copy; later, when attempting to retrieve the key, we should lookup these replica places and the lookup can be successful provided that there is at least one copy alive. During implementation, the process of replication and retrieval may be different for different DHTs, e.g., we can use the successor nodes to place the replicas; and it is also possible to use hashing to determine replica places.



(a) Exponential, 1 hour (b) Exponential, 10 hours (c) Pareto, 1 hour (d) Pareto, 10 hours

Figure 4: Expected lifetime of a replicated key. (a) Exponential lifetime distribution with $E[L] = 1$ hour; (b) Exponential lifetime distribution with $E[L] = 10$ hours; (c) Pareto lifetime distribution with $E[L] = 1$ hour; (d) Pareto lifetime distribution with $E[L] = 10$ hours.

However, no repair will be performed for the failed replica, and the whole replica system dies when all the replicas fail. In that situation, the following lookups for the replicated data key fail accordingly.

Here, we adopt the lifetime of a replicated key T, instead of the reliability probability (i.e., "number of nines"), as the metric to evaluate the key reliability. Given the peer lifetime distribution, we can deduce the lifetime of the replicated key accordingly.

Two typical peer lifetime distributions are considered here: (i) ***Exponential distribution***, which is widely used in reliability theory to model the time to failure of components. Its CDF is given by $F(x) = 1 - e^{-\lambda x}, (\lambda > 0)$; and (ii) ***Pareto distribution***, which is observed in real P2P systems [11], with the CDF given by $F(x) = 1 - (1 + \frac{x}{\beta})^{-\alpha}$, where α represents the heavy-tailed degree and β is a scale parameter. Under Pareto distribution, most peers have a short lifetime, while a few peers stay much longer in the system.

For a replicated key without repair, we have the following theorem:

Theorem 6 *Given the replication factor r, the expected lifetime of a replicated key without repair is given by*

$$E[T] = \sum_{i=1}^{r} \binom{r}{i} \frac{(-1)^{(i+1)}}{\lambda i}$$

for exponential lifetime distribution and

$$E[T] = \sum_{i=1}^{r} \binom{r}{i} (-1)^i \frac{\beta}{i(1-\alpha)+1}$$

for Pareto lifetime distribution with $\alpha > 2$.

3.4.2 Replication with repair mechanism

The replication process is similar to the above, but a repair mechanism is installed. Periodically, some responsible node (maybe the key owner, or the replica) checks the liveness of other replicas. In case that one replica is found to be dead, the responsible node will perform repair by recopying the data key to a new node. By this means, we keep the number of replicas unchanged; however, if the responsible node is dead, no repair will be performed any longer. The repair rate is $\mu = \frac{1}{T_{rp}}$, where T_{rp} is the period between two consecutive repairs performed by the responsible node. For a replicated key with repair, our results are as follows:

Theorem 7 *Given the replication factor r and the repair rate μ, the expected lifetime of a replicated key with repair is given by*

$$E[T] = \frac{1}{\Pi_{k=1}^{r-1} \frac{k\theta}{k\theta+\mu}} [\frac{1}{r\theta} + (1 - \Pi_{k=1}^{r-1} \frac{k\theta}{k\theta+\mu}) \frac{1}{\mu}]$$

where $\theta = \lambda$ for exponential lifetime distribution and $\theta = (\alpha - 2)/\beta$ for Pareto lifetime distribution with $\alpha > 2$.

In Fig. 4, we compare two kinds of replica policies under exponential lifetime distribution and Pareto lifetime distribution respectively. For each distribution, we consider two lifetime expectations, i.e., $E[L] = 1$ hour and $E[L] = 10$ hours. The former has a high churn rate, while the later has a low churn rate. For a given lifetime expectation, exponential distribution has a fixed λ; but for Pareto distribution, α, β are not fixed. In our study, we consider different heavy-tailed degrees by varying the value of α.

From Fig. 4, we can observe that, under exponential lifetime distribution, it is hard to achieve a long lifetime simply by replication. Without a repair mechanism, even with 30 replicas, the expected lifetime of the replicated key is still rather short even under a low churn environment (i.e., $E[L] = 10$ hours). However, the situation is a bit different for Pareto lifetime distribution. It can be found that, the repair mechanism is not indispensable for Pareto lifetime distribution with a very heavy tail. Only by replication, we can achieve a rather long lifetime for the replicated key. It is true

even in a high-churn environment. For instance, in the parameter setting (i.e., $E[L] = 1$ hour, $\alpha = 2.01, \beta = 1.01$), without repair mechanism, by using 15 replicas, the expected lifetime of the replicated key is longer than 61 days. But if we want to increase the lifetime of replicated key to the level of years, it is necessary to install a repair mechanism. With the linear increase of repair rate, the lifetime is increased almost exponentially.

As to the selection of the replication factor and repair rate, it is application-specific. We should consider some practical issues during design, such as data type, data lifetime, repair cost, etc. The data stored under the $< key, data >$ pair can be a data fragment (e.g., in storage systems), a file pointer (e.g., in file-sharing systems), or a metadata item (e.g., in information management systems). Also the data itself has a lifetime and it is not always necessary to keep the key as long as possible. For example, in file-sharing systems, in case the node leaves the system, it becomes unnecessary to continue the repair of the DHT keys published by that node.

We still use the Kad file-sharing system as an example. The Kad algorithm produces 19 key replicas on average, and the file holder republishes the key every 5 hours. The key becomes expired if no refresh is performed. We compare the lifetime of the replicated key under two different replication policies (in Table. 2) based on Theorem 6 and 7. From the results, we find that even without repair, Kad still can guarantee the existence of the key about 17.5 days ($\gg$ peer lifetime expectation $E[L] = 2.71hr$). Although the lifetime of the replicated key without repair is not long enough, it is safe for us to perform repair lazily. When equipped with a lazy republishing mechanism, the lifetime is further improved to the level of years. It means that, if the file holder keeps staying the system and performs republishing periodically, the lookups towards that key can be successful with very high probability.

Replication Policy	Expected Lifetime
without repair	17.5 days
with repair ($T_{rp} = 5$ hours)	19873 days

Table 2: Expected lifetime of replicated key

In other scenarios, given the information about the peer lifetime distribution, Theorems 6 and 7 can give us great help in choosing proper parameters.

4 Conclusion

In this paper, our objective is to provide directions for DHT design under churn through analytical study. We consider three important aspects in handling churn for DHT systems. For lookup strategy, two typical lookup strategies - *Recursive Routing* and *Iterative Routing* - are studied in terms of their latency and overhead. We show their effectiveness under different system settings and explore the existence of their enhancements. Then, we study the performance of parallel lookup and the selection of the degree of parallelism. Later, we analyze two key replication policies to handle data key loss due to churn, and discuss whether a repair mechanism is necessary and how to select the replication factor. In our future work, we plan to investigate more design aspects on improving DHT performance.

References

[1] http://www.emule-project.net/.

[2] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proc. of NSDI'04*, 2004.

[3] S. El-Ansary, S. Krishnamurthy, E. Aurell, and S. Haridi. An analytical study of consistency and performance of dhts under churn. In *SICS Technical Report T2004:12. Swedish Institute of Computer Science, Stockholm, Sweden. ISSN 1100-3154*, 2004.

[4] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi. A statistical theory of chord under churn. In *Proc. of IPTPS'05*, 2005.

[5] S. S. Lam and H. Liu. Failure recovery for structured p2p networks: Protocol design and performance evaluation. In *Proceedings ACM SIGMETRICS*, 2004.

[6] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek. Bandwidth-efficient management of dht routing tables. In *Proc. IEEE NSDI'05*, 2005.

[7] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *proceedings of IEEE INFOCOM*, 2005.

[8] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *ACM Conf. on Principles of Distributed Computing (PODC)*, 2002.

[9] N. Prabhu. *Stochastic processes; basic theory and its applications*. New York : Macmillan, 1965.

[10] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proc. of USENIX Annual Technical Conf.*, 2004.

[11] S. Saroiu. Measurement and analysis of internet content delivery systems. *Doctoral Dissertation, University of Washington*, Dec. 2004.

[12] D. Stutzbach and R. Rejaie. Characterizing churn in peer-to-peer networks. In *Technical Report CIS-TR-05-03, Univ. of Oregon*, 2005.

[13] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed dht. In *proceedings of IEEE INFOCOM*, 2006.

[14] D. Wu, Y. Tian, and K. W. Ng. Analytical study on improving lookup performance of distributed hash table systems under churn. In *CUHK Technical Report, http://www.cse.cuhk.edu.hk/∼dwu/files/dht-tr.pdf*, 2006.