Survive Under High Churn in Structured P2P Systems: Evaluation and Strategy

Zhiyu Liu, Ruifeng Yuan, Zhenhua Li, Hongxing Li, and Guihai Chen*

State Key Laboratory of Novel Software Technology,
Nanjing University, China
gchen@nju.edu.cn

Abstract. In Peer to Peer (P2P) systems, peers can join and leave the network whenever they want. Such "freedom" causes unpredictable network environment which leads to the most complex design challenge of a p2p protocol: how to make p2p service available under churn? What is more, where is the extreme of a system's resistibility to high churn? A careful evaluation of some typical peer-to-peer networks will contribute a lot to choosing, using and designing a certain kind of protocol in special applications. In this paper we analyze the performance of Chord [1], Tapestry [2], Kelips [3], Kademlia [4] and Koorde [5], then find out the crash point [6] of each network based on the simulation experiment. Finally, we propose a novel way to help nodes survive under high churn.

Keywords: Peer-to-Peer, Fault Resilience, High Churn, Crash Point.

1 Introduction

People like to use peer-to-peer networks, because there are few restrictions. Peers can join and leave the network whenever they want. However, such "freedom" causes unpredictable network environment which leads to the most complex design challenge of P2P protocols: how to make p2p service available under churn? Almost every P2P protocol has proposed its method to deal with churn, and shows some experiment report. However, no previous work has compared those protocols in the aspect of "resilience under high churn".

Why is the problem "resilience under high churn" important? Before answering the question, let us give "churn" and "high churn" a descriptive definition.

Ordinary churn: is such a condition that nodes join the network one by one, or leave gracefully by informing their neighbors. The churn event(ie, node join and departure) happens occasionally and can be handled quickly so we can suppose that the overlay of the network is well structured before any individual churn event occurs.

High churn: is such a condition that large percent of nodes join and/or silently leave the network simultaneously and frequently.

* Corresponding author.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, pp. 404–411, 2006.

From above we can see that: first, high churn condition is totally different from ordinary churn: the stabilization routine which may work very well under ordinary churn could be useless under high churn. Thus, good performance under churn does not imply the same result under high churn. We have to find other ways to make sure the service is available under high churn.

Second, high churn is not just an imagination but does happen from time to time in real life. For example, a large number of maliciously controlled peers could leave the network simultaneously; the power is cut off over a wide area; temporarily hot resources like "world cup online show" may also cause a large number of peers to join and leave simultaneously.

Third, by taking a look at "resilience under high churn" we can evaluate a P2P protocol in more comprehensive way. Choosing a P2P network with better "resilience under high churn" will help us survive in extremely turbulent environment.

A lot of related works addressing to "resilience" problem have been done. Some [7, 10, 11, 14] did excellent theoretical analysis and some did carefully selected simulations [11]. So far, however, simulation of peer-to-peer systems under high churn to compare their "resilience under high churn" has not been done yet. In this paper, we first propose a measurement of "resilience under high churn" of P2P protocols: crash point, then fairly evaluate some typical P2P protocols like Chord [1], Tapestry [2], Kelips [3], Kademlia [4], Koorde [5] and so on to demonstrate their different performances of "resilience under high churn" by "crash point", and finally design a strategy to help live node survive under high churn. Here is the definition of crash point:

Crash point. If x percent of nodes' leaving simultaneously causes half (50%) randomly generated look-ups to fail, then x percent is defined as crash point.

Crash point is so defined for three reasons:

1. Both concurrent joining and leaving lead to the incorrect routing information, so without loss of generality, the percentage of concurrent leaving nodes represents the degree of churn.
2. By this definition, we can ignore the difference between two kinds of crash. One kind is that when a node becomes isolated, all look-ups from/to it will fail. The other kind is that when the whole network breaks up into some disconnected sub-nets, look-ups between nets will fail. However, LOOK-ups within a sub-net can still succeed. Then we can compare different protocols under the same criterion.
3. Successful look-up ratio is easy to record and it has certain relationship with the connectivity of the network. We discover in the simulation experiments that once half of the look-ups fail, the network could never be recovered to fully connected status.

The rest of the paper is organized as follows: section 2 introduces some related works. Section 3 discusses some important factors of a DHT structure which have impact on "crash point". Section 4 shows the experiment results.

Section 5 gives a description of the strategy DARE (Detect Automatically and Rejoin Efficiently) to help nodes survive under high churn. Finally section 6 concludes the paper and points out the future work.

2 Related Work

Liben-Nowell *et al.* [7] examine error resilience dynamics of Chord when nodes join/leave the system and derive lower bounds on the degree necessary to maintain a connected graph with high probability. Fiat *et al.* [8] build a Censorship Resistant Network that can tolerate massive adversarial node failures and random object deletions. Saia *et al.* [9] create another highly fault-resilient structure with $O(\log 3N)$ state at each node and $O(\log 3N)$ hops per message routing overhead.

Unfortunately, very few studies examine the resilience of existing graphs in comparison with each other, especially when nodes join/leave at a high rate. We are aware of only few comparison study, including that Gummadi *et al.* [11] find that ring-based graphs (such as Chord) offer more flexibility with route selection and provide the best resilience performance compared with some other DHTs routing algorithms, however they did not mention some other good DHTs like Kelips which shows very good experiment result in our simulations, Dmitri Loguinov *et al.* [14] do some theoretical analysis of the existing graphs, Jinyang *et al.* [12] compare several DHTs under churn in the aspect of look-up latency, and Simon *et al* [13] address the question of how high a rate of node dynamics can be supported by structured P2P networks, however they confine their study to hypercube only.

We began this work from 2004, and showed the resilience of ring topology of P2P overlay in [6]. Now we give more evidences to support the rationality of "crash point" and compare more topologies of structures other than ring, and finally give a strategy to help nodes survive better under high churn.

3 Analysis

One of the reasons DHTs are seen as an excellent platform for large scale distributed systems is that they are resilient in the presence of node failure. This resilience has two different aspects [11]:

Static resilience. We keep the routing table static, only delete the items of failed nodes to see whether the DHTs can route correctly without the help of stabilization routine.

Routing recovery. They repopulate the routing table with live nodes, deleting the items of failed nodes.

3.1 Static Resilience

Gummadi *et al.* [11] has concluded that flexibility is the most important factor that affects the performance of static resilience, and our simulation result is quite

supportive to that conclusion. When basic routing geometry has been chosen, more flexibility means more freedom in the selection of neighbors and routes. Two cases are discussed respectively as follows:

Neighbor Selection. DHTs have a routing table comprised of neighbors. Some algorithms make purely deterministic neighbors (i.e., Koorde), others allow some freedom to choose neighbors based on other criterias in addition to the identifiers; most notably, latencies have been used to select neighbors. (i.e., Tapestry).

Route Selection. Given a set of neighbors, and a destination, the routing algorithm determines the choice of the next hop. However, when the determined next-hop is down, flexibility will describe how many other options are there for the next-hop. If there are none, or only a few, then the routing algorithm is likely to fail.

Chord, Kademlia, Klips provide both neighbor selection flexibility and routing selection flexibility. While Tapestry only provides neighbor selection flexibility, and Koorde has no flexibility in either neighbor selection or route selection. We can see their difference from simulation results present in the next section.

3.2 Routing Recovery

Three kinds of routing recovery strategies are usually used.

On demand recovery. This kind of recovery happens whenever outside environment asks the node to change. For instance, a neighbor informs you its departure, then you delete it from your neighbor list and replace it with a new neighbor.

Stabilization routine. This kind of process actively runs every certain period to eliminate the error in routing table.

Piggybacked recovery. Some protocols can use incoming messages like "look-up request" to recover the routing table if necessary.

On demand recovery is useful and efficient under ordinary churn. However, it becomes useless and inefficient under high churn. On the other side, stabilization routine plays an important role under high churn. Although different stabilization policies are adopted by different DHTs, we adjust the parameters to let all the protocols run stabilization routine at the same frequency. While Kademlia, Kelips can use piggybacked recovery owing to their symmetric routing path, however Chord, Koorde, and Tapestry have no such routing recovery strategy because their routing path is asymmetric.

4 Simulation Results

In all the following simulations, we use a network of $N = 1000$ nodes, with average `round_trip_time`$(RTT) = 2s$ ($1s$ equals to 1000 simulation time units: ms) between any pair of nodes. Each node generates look-up requests at the

interval exponentially distributed about the mean time of $10s$, and the look-up requests are for randomly selected keys. At time $1800s$, when the topology is supposed to be stable, we let a portion (from 0% to 70%, increasing 10% each time) of nodes leave the network simultaneously. Stabilization routine is trigged every 100s on average. During the simulation we record the average successful look-up ratio (SucRatio) every 30s. If a look-up does not reach the destination in 20s ($20 \approx \log_2 N * RTT$), we consider it as a failed look-up. Each simulation test case is repeated 5 times to avoid randomerror. All the simulations are run on p2psim [15].

For the rest of this section, we address two questions:

Question 1: *Does the crash point really mean something?*

We first explain why to choose 50% successful look-up ratio (SucRatio) as a sign of crash. SucRatio is easy to obtain and compute, and it has certain relationship between the degree of network connectivity. If a network crashes down, it will finally have impact on SucRatio. Let us suppose the network of size N is broken up into two equal-sized networks of size $N/2$, then, the look-ups within the subnetwork ($\frac{N}{2} \times \frac{N}{2}$) will succeed, and look-ups aimed to the other sub-network will fail. In average,

$$SucRatio = \frac{\frac{N}{2} \times \frac{N}{2} \times 2}{N \times N} \times 100\% = 50\%.$$

If SucRatio is much higher than 50%, we can expect that the network is still fully connected, but if SucRatio is much lower than 50%, the network is probably turned into pieces, thus the crash moment is not far.

Our assumption of crash sign (i.e., 50% SucRatio) is further verified by the simulation on Tapestry. Fig. 1 is the simulation result of Tapestry, it shows clearly that when 50% percent of nodes fail, the SucRatio reduces to 50% , and the network never recovers from the failure. When failed nodes are less than 40%, the network recovers very well. 50% is our best choice, however, you can change the percentage to redefine the "crash point" as long as it suits the application.

Question 2: *How do the crash points of various protocols compare?*

Here we choose five typical protocols: *Chord*, which is the representative of ring structure; *Tapestry*, which is the representative of tree structure; *Kademlia*, which implements an XOR structure; *Koorde*, which embeds a De Burijn graph into a ring structure, and *Kelips*, which claims to sacrifice memory for better lookup latency and resilience. In each protocol, every node is configured to have about 20 neighbors at the beginning, except Kelips-node has about 60 neighbors which are the least amount owing to its special join strategy.

Fig. 2 shows the successful look-up ratio (SucRatio) for varying percentage of nodes failures across different protocols. We can see that, although Kelips has much more neighbors, it does not do better than Chord. Kelips's crash point is around 70%, and Chord's crash point is behind 70% (From [11, 6] we can tell the crash point of Chord is around 80%). Tapestry has more neighbor selection flexibility than Koorde, while Koorde is based on a ring which is improved to have a better resilience than tree structure, so they have a close performance,

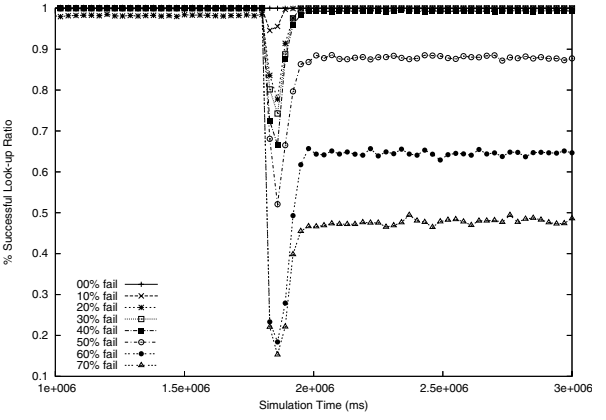

Fig. 1. Tapestry: Successful look-ups ratio for varying percentage of node failures

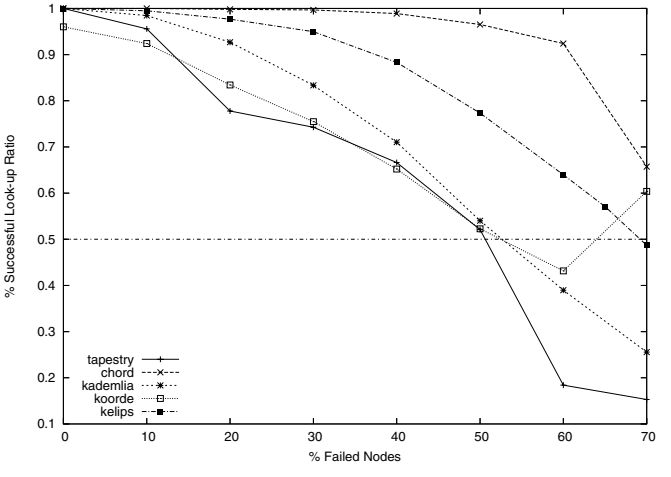

Fig. 2. Successful look-up ratio for varying percentage of node failures across different protocols

both have crash point around 50%. In comparison of Tapestry with Chord, we can conclude that flexibility in routing selection is more important than neighbor selection to improve the resilience. Kademlia is in the middle lever, however we can expect it to perform better in a larger scale network owing to its piggybacked recovery strategy. We notice that for Koorde SucRatio with 70% failure nodes is higher than that with 60%. That is because when a large percentage of nodes fail, there are much fewer look-ups generated, and some of them can quickly reach their destinations without suffering a long delivering path. However since the look-ups are very few, static method became inaccurate, that is why we stop the simulation with node failure portion at 70%.

5 Survive Strategy — A Use of Crash Point

In this section, we address this question: *what a live node should do to survive under high churn or how to rescue itself?*

We can see from the simulation that once the degree of churn exceeds the crash point of a network, the network is probably torn to pieces. The routing recovery strategy could not reunite the sub-networks into a whole all by itself. Thus we here propose a strategy — Detect Automatically and Rejoin Efficiently (DARE) — to partially solve the problem. We separate DARE into three phases:

Phase 1: Detecting. Peer in this phase do nothing but record and watch over the look-up success ratio. Once the look-up success ratio is lower than 50%, the node meets the crash point, and it will enter phase 2.

Phase 2: Electing. Peers entering this phase are in danger of isolation, they should try to rejoin from the "well-known" node(s) in order to unite again. However, aimed to reduce the overhead of rejoin, a representative in each sub-net will be elected to enter phase 3. If it succeeds, there is no need for the rest of its sub-net to rejoin. They could be recovered later by stabilization routine. Some classical election algorithm like the bully algorithm [16] could be adopted here to ensure one and only one representative is select out. However, if we loosen the restriction and let more than one representative exist in a neighborhood, it is still acceptable.

Phase 3: Rejoining. Peers in this phase are the representatives of their neighborhoods. Since they are separated from other peers, there is only one way to rejoin the "big family" — rejoin from the "well-known " node(s). If it succeeds, all of its neighbors are rescued, otherwise, it have 2 choices: waiting for other neighbors to rescue it or informing its upper user of the crash of the network.

DARE has several advantages. First, it can help nodes to discover the crash status automatically because we set a warning line according to "crash point". Second, nodes which notice the danger of crash can take self-rescue as early as possible. Third, only one node in the neighborhood will trigger the rejoin process, which can avoid a large number of concurrent joins and reduce the workload of well-known node(s).

6 Conclusion and Future Work

In this paper, we propose a measurement of "crash point" to compare the "resilience under high churn" across some typical structured P2P protocols, and present a strategy to help the alive nodes survive under high churn. It shows that Chord with ring topology has the best resilience under high churn; crash point is sensitive to flexibility in routing selection but not very sensitive to the amount of neighbors.

In the future work, we will study more about the relationship between SucRatio and the connectivity of the network. We will also develop DARE to show

its performance, hoping to find a better way to discover the danger of network disconnection, and prevent the network to be separated.

Acknowledgement

The work is partly supported by China NSF grant (No. 60573131), Jiangsu Provincial NSF grant (No. BK2005208), China 973 project (No. 2002CB312002), and TRAPOYT award of China Ministry of Education. We also thank Jun Li, Wentao Zheng and all the other reviewers for their valuable suggestions.

References

1. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM (2001)
2. B.Y. Zhao, J.Kubiatowicz, A. D. Joseph: Tapestry: a fault-tolerant wide-area application infrastructure. Computer Communication Review 32(1): 81 (2002)
3. I. Gupta, K. P. Birman, P. Linga, A. J. Demers, R. van Renesse: Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. IPTPS (2003)
4. P. Maymounkov and D. Mazieres: Kademlia: A peerto -peer information system based on the xor metric. In Proceedings of IPTPS '02,(2002)
5. M. F. Kaashoek and R. Karger: Koorde: A simple degreeoptpimal distributed hash table. In 2nd International workshop on P2P Systems(IPTPS03),(2003)
6. Z. Liu, G. Chen, C. Yuan, S. Lu, C.Xu: Fault Resilience of Structured P2P Systems. WISE (2004)
7. D. Liben-Nowell, H. Balakrishnan, and D. Karger: Analysis of the Evolution of Peer-to-Peer Networks. ACM PODC (2002)
8. A. Fiat and J. Saia: Censorship Resistant Peer-to-Peer Content Addressable Networks. ACM/SIAM Symposium on Discrete Algorithms (2002).
9. J. Saia, A. Fiat, S. Gribble, A.R. Karlin, and S. Saroiu: Dynamically Fault-Tolerant Content Addressable Networks. IPTPS (2002).
10. J. Aspnes, Z. Diamadi, and G. Shah: Fault-Tolerant Routing in Peer-to-Peer Systems. ACM PODC (2002)
11. K.P. Gummadi, R. Gummadi, S.D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica: The Impact of DHT Routing Geometry on Resilience and Proximity. ACM SIGCOMM (2003)
12. J. Li, J. Stribling, T. Gil, R. Morris, F. Kaashoek: Comparing the performance of distributed hash tables under churn. IPTPS (2004)
13. S.S.Lam and Huaiyu Liu: Failure Recovery for Structured P2P Networks: Protocol Design and Performance Evaluation. ACM SIGMETRICS/Performance '04 (2004)
14. D. Loguinov, A. Kumar, V. Rai, S. Ganesh: Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. ACM SIGCOMM (2003)
15. http://pdos.lcs.mit.edu/p2psim/howto.html
16. Garcia-Molina, H.: Elections in a Distributed Computing System. IEEE Transactions on Computers, Vol. C-31, no. 1, (January),(1982) pp. 48-59.