Optimizing Routing in Structured Peer-to-Peer Overlay Networks Using Routing Table Redundancy

Rongmei Zhang and Y. Charlie Hu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907

Peter Druschel
Department of Computer Science
Rice University
Houston, TX 77005

1 Introduction

Structured peer-to-peer (p2p) overlay networks like CAN, Chord, Pastry and Tapestry [3, 6, 5, 9] provide a self-organizing substrate for large-scale peer-to-peer applications. These systems provide efficient, fault-tolerant routing, object location and load balancing within a self-organization overlay network.

In this paper, we show how redundant information that is collected as part of the normal overlay maintenance protocol can be exploited to improve the performance of routing, in terms of both the number of routing hops and routing delay penalty. We use Pastry as a concrete example to describe the set of optimizations and to evaluate their improvement in routing performance via a large scale simulation using a realistic network topology model. We then discuss how these optimizations can be applied to other structured p2p overlays.

2 Background on Pastry

In this section, we give a brief description of Pastry. Pastry is a scalable, fault-tolerant, peer-to-peer substrate. Each Pastry node has a unique, uniform randomly assigned *nodeId* in a circular 128-bit identifier space. Given a 128-bit key, Pastry routes the associated message towards the live node whose nodeId is numerically closest to the key.

For the purpose of routing, nodeIds and keys are thought of as a sequence of digits in base 2^b (b is a configuration parameter with typical value 4). A node's routing table is organized into $128/b$ rows and 2^b columns. The 2^b entries in row n of the routing table contain the IP addresses of nodes whose nodeIds share the first n digits with the present node's nodeId; the $n+1$th nodeId digit of the node in column m of row n equals m. A routing table entry is left empty if no node with the appropriate nodeId prefix is known. The uniform random distribution of nodeIds ensures an even population of the nodeId space; thus, on average only $\lceil log_{2^b} N \rceil$ levels are populated in the routing table.

Each node also maintains a *leaf set*. The leaf set is the set of l nodes with nodeIds that are numerically closest to the present node's nodeId, with $l/2$ larger and $l/2$ smaller nodeIds than the current node's id. A typical value for l is approximately $\lceil 8 * log_{16} N \rceil$. The leaf set ensures reliable message delivery and is used to store replicas of application objects.

At each routing step, a node seeks to forward the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the current node's shared prefix. If no such node can be found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id. Several such nodes can normally be found in the routing table; moreover, such a node is guaranteed to exist in the leaf set unless the message has already arrived at the node with numerically closest nodeId or its immediate neighbor. And, unless all $l/2$ nodes in one half of the leaf set have failed simultaneously, at least one of those nodes must be live.

Experiments and analysis [5, 2] show that the expected number of forwarding hops is slightly below $\lceil log_{2^b} N \rceil$, with a distribution that is tight around the mean.

Node joining An arriving node with new nodeId X joins the network by asking an existing, nearby Pastry node A to route a special message using X as the key. (The IP address of an existing Pastry node must be learnt through some out-of-band mechanism). The message is routed to the existing node Z with nodeId numerically closest to X. X then obtains the leaf set from Z and appropriate routing table entries from nodes encountered along path from A to Z. Using this information, X can correctly initialize its state and notify other nodes that need to know of its arrival, thereby updating their node states.

Locality-aware routing Pastry maintains a proximity-aware overlay by minimizing the distance, according to a proximity metric like network delay, to each of the nodes that appear in a node's routing table. Each entry refers to one of the nearest nodes with the required prefix match. Since the expected number of nodes with a given nodeId prefix decreases exponentially with the length of the prefix match, the expected distance to an entry in a node's routing table increases exponentially with the length of the prefix match. As a result, in routing a random message, the expected distance traveled in each successive routing step increases exponentially, and the distance of the last step dominates. Because of this property of prefix routing, Pastry routing exhibits two locality properties: (1) *Low delay stretch*: Analysis and simulations in [2] using realistic network topologies shows that the total delay experienced by Pastry routing relative to the delay between the source and destination via the underlying IP routing is usually below two; (2) *Local route convergence*: Analysis and simulations also show that the routes of messages sent from nearby nodes to the same destination tend to converge at a node near the sending nodes.

2.1 Redundancy in routing tables

Depending on how Pastry is implemented, each routing table entry can hold more than one nodes. In [2], routing table maintenance was proposed to prevent the deterioration of routing quality when the proximity metric may change dynamically. For each row in the routing table, the maintenance procedure periodically requests the corresponding row from a randomly chosen entry in that row. Each entry in the obtained row is compared with the current entry and the closest node is installed in the routing table. When a node is replaced during the routing table maintenance or as a result of an update caused by the arrival of a new node, it is kept in a list of alternative nodes instead of being removed from the routing table. The nodes in the list are selected based on proximity if there are more candidates than the size of the entry. In [2], up to 10 nodes are saved in each entry of the routing table. Effectively, the routing table becomes 3-dimensional (3-D) with a vector of candidates, called the *route set*, in each entry.

The redundancy of routing tables was originally introduced for improved fault tolerance, but in this paper we are only interested in its potential benefits of improving the routing performance in terms of the routing delay and the number of p2p routing hops. Therefore, we do not use the redundancy for recovery from routing failures when node failures are present. If the primary node from the entry that shares the longest prefix with the message key is found to have failed, instead of falling back on one of the backup nodes, Pastry chooses the node from other entries that is numerically closest to the message key, and again only primary nodes are considered. The purpose of this decision is to seperate the effects of fault tolerance from the effects of the optimizations and to have a better understanding of the optimizations.

3 Optimizing routing using routing table redundancy

In addition to its use for routing table maintenance and failure recovery, the 3-D routing table also provides opportunities for improving routing performance, in terms of both delay penalty and the number of routing hops. In the following, we present two optimizations that take advantage of the redundancy in the 3-D routing tables. We conduct experiments to evaluate the optimizations via a large scale simulation using a realistic network topology model. The impacts of possible node failures in the p2p network are also considered.

3.1 Experimental setup

Our experiments in this section are performed via simulations on a network topology with 1050 routers, which is generated by the Georgia Tech random graph generator using the transit-stub model [8]. There are 5 transit domains, with 10 routers in each. Each transit router has 5 stub domains attached, and each stub has 4 routers. The routers are not part of the overlay networks. Instead, the overlay networks are formed by 20,000 end nodes that are randomly assigned to the 1000 routers in the stub domains with uniform probability. The routing policy weights generated by the Georgia Tech random graph generator are used to perform IP unicast routing in the IP network.

To evaluate the performance of the optimizations when nodes in the p2p network may fail and come back, we use a trace of node arrivals and departures from a study measuring the availability of desktop computers in a corporate network [1]. The original trace contains the liveness states of 65,000 nodes over a period of 840 hours. Our simulations use 20,000 nodes during the first 60 hours. Out of the 20,000 nodes, on average there are 260 nodes that fail (or leave the network) and 290 nodes that join (or re-join) the network within each of the 60 hours. The average failure rate is about 10% and on average there are about 16,600 live nodes in the network during a given hour. We assume that nodes fail silently and the failed node is not discovered by another node until that node tries to route through it a message, either a data packet, or a control message as part of the routing table maintenance.

During each hour in the simulation, 500 messages are sent with random starting nodes and random keys. We measure the number of routing hops in the Pastry overlay network, and the routing delay penalty, averaged over the total of 30,000 messages (i.e., over 60 hours). The *routing delay penalty* is defined as the ratio between the distance traversed in the Pastry route and the distance traversed if it were routed directly by the underlying IP network. If the message is forwarded to a node that has failed in the simulation, the round trip delay to the failed node is added to the total delay experienced by the message to count for the effects of the node failure.

The routing table maintenance procedure as described in previous section is enabled for the simulations with and without node failures. When nodes fail, a process called leaf set maintenance [5] is used to restore the consistency of leaf sets when node failures are discovered in the leaf set. Leaf set maintenance is also performed periodically as is routing table maintenance at each live node in the Pastry network. We compare the results with and without periodic maintenance procedures. The maintenance frequency varies from every 2 hour to every 16 hours when the periodic maintenance is enabled.

3.2 Optimization 1: searching the destination node in the backup list

Pastry routing normally does not consider backup entries in the routing table when forwarding messages. There is, however, a possibility that the destination is not the nearest node in the appropriate routing table entry, but is a member of that entry's backup list. Thus, its IP address could be looked up and the message could be forwarded directly to the destination. In this case, the message can be delivered in one more hop, and at least one (p2p) routing hop is saved. As a result, the routing delay will be reduced in most cases. This is because the triangle inequality holds in most cases in the Internet: the direct IP route between two nodes A and B is usually shorter than going through a third node, e.g., from A to C, followed by C to B.

When there is a match between the message key and a node in the routing table entry but the node is later found to have failed, Pastry will use the original 2-D routing algorithm, which ignores all the backup entries. For example, it will first try the primary node in the route set that shares the longest prefix with the message key. If no such entry is found, it will look for the best alternative whose nodeId is closest numerically to the message key, and again, only primary nodes are considered.

We denote the above optimization as "exact-match". Clearly, it is effective only when the destination key of the message is equal to the nodeId of some live node in the system. Figures 1 and 2 show that both the routing delay penalty and the average number of routing hops are reduced significantly with this optimization when the message key is the nodeId of a random node that exists in the network.

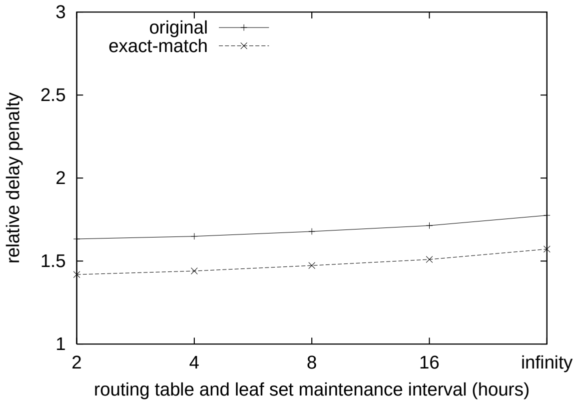

Figure 1. Relative delay penalty for original and improved Pastry routing when destination is a random existing nodeId. The x-axis shows the delay between two consecutive routing table maintenance.

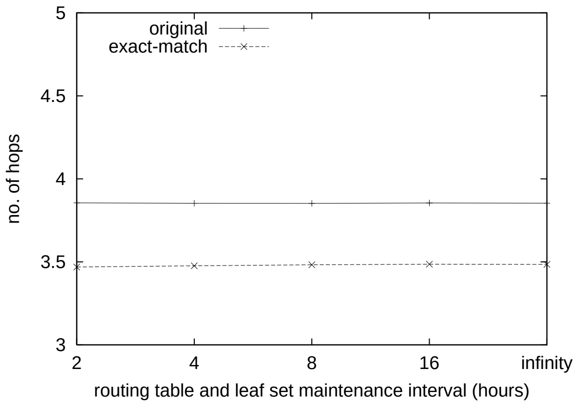

Figure 2. Average number of routing hops for original and improved Pastry routing when destination is a random existing nodeId.

Specifically, the average number of routing hops is reduced by about 10%, and the routing delay penalty is reduced by about 13% (relative to the IP routing).

Figure 3 and 4 show the results with node failures based on the availability trace data. The average number of routing hops is reduced by 7%-10%, and the average routing delay penalty is reduced by about 11%-14%. Note that routing table maintenance and leaf set maintenance can reduce the impacts of node failures by replacing failed nodes in routing tables and leaf sets, and the higher the maintenance frequency, the better the performance of Pastry routing. On the other hand, the maintenance procedures and their frequency have little effects on the routing performance when the p2p network is failure free and the underlying network is static.

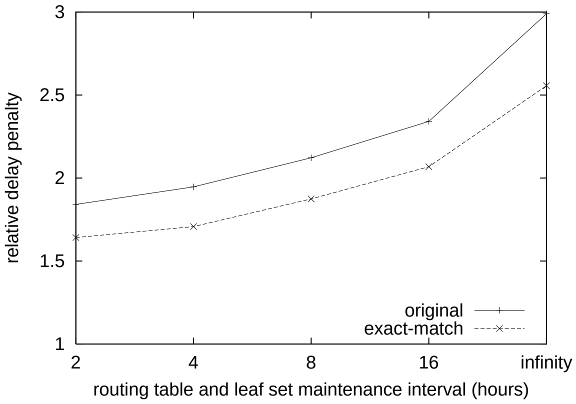

Figure 3. Relative delay penalty for original and improved Pastry routing with node failures, when destination is a random existing nodeId.

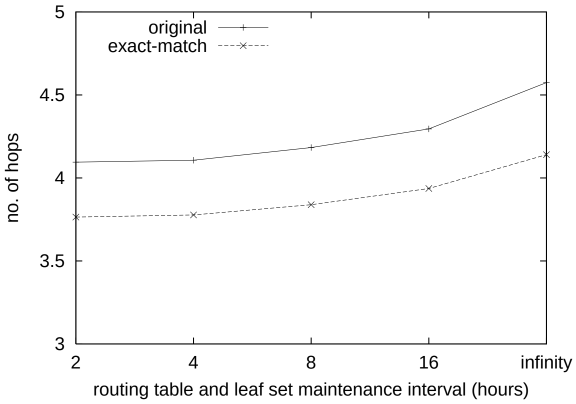

Figure 4. Average number of routing hops for original and improved Pastry routing with node failures, when destination is a random existing nodeId.

3.3 Optimization 2: probabilistically choosing the next hop from the backup list

A more aggressive approach to utilizing the redundancy in Pastry routing tables is to choose the next hop based on the probability that a routing table entry is the destination of the message (i.e., the live node with nodeId numerically closest to the key). This probability can be estimated based on the density of the nodeId space, e.g., the distance between the nodeIds of two adjacent nodes in the id space, as observed in a node's leaf set.

At each step of routing, if the destination key is not within the leaf set nor does it match any nodes in the corresponding route set, there is still a certain possibility that the destination is one of the nodes in the route set, if the destination key and that nodeId are close enough in the id space. To exploit this possibility, the routing is modified so that whenever the distance between the destination key and the nodeId of some node in the route set is smaller than c times the average distance between any two neighboring nodes in the id space, that closest candidate is taken as the next hop. To approximate the average distance in the id space between any two neighboring nodes in the network, each node simply uses the average distance between adjacent nodes in its leaf set.

The parameter c is a constant whose choice controls the aggressiveness of the guess. When the closest candidate is beyond the threshold, the routing falls back to using the closest candidate in the proximity space. Obviously this approach is probabilistic, since it is possible that the guess is wrong and the message has to be forwarded further at the next hop. We denote this optimization "probabilistic". Note that even if the guess is wrong, and further hops are needed, taking the closest node in the id space makes better progress towards the final destination in the id space, compared to taking the node in the same routing table entry that is closest in the proximity space.

As in the case of "exact-match", Pastry also falls back on the original routing algorithm if the next hop chosen by the probabilistic optimization turns out to have failed.

We experimentally compare the delay stretch and routing hops by using the original Pastry routing, the exact-match optimization, and the probabilistic optimization, varying the choice of c, and the results are shown in Figures 5 and 6. Unlike in Figures 1 and 2, the destination of each message is chosen as a random key in the id space; given the number of nodes in the network, a message key is highly unlikely to coincide with any existing nodeId in the network. The simulations confirm that the exact-match optimization presented in the previous subsection does not have any noticeable effect on the performance of Pastry routing when the destination key is chosen randomly. However, the probabilistic optimization can improve both the relative delay penalty and the routing hops significantly.

For the probabilistic method, there is a tradeoff in the choice of the threshold between reducing the per-hop routing distance versus reducing the number of routing hops, as follows. The larger the threshold, the more likely the routing will take the-closest-in-id-space hops; while the smaller the threshold, the more likely the routing will take the-closest-in-proximity-space hops. We experientially measure the routing performance for a set of choices, and the results are shown in Figures 5-6. The results show that the larger the value of c is, the fewer the average number of hops taken. Setting the threshold to infinity minimizes the number of routing hops. Setting the threshold to be 1.0–1.1 times the average distance between adjacent nodeIds in the leafset minimizes the delay penalty. In this case, the delay penalty is between 6%–8% lower than Pastry without this

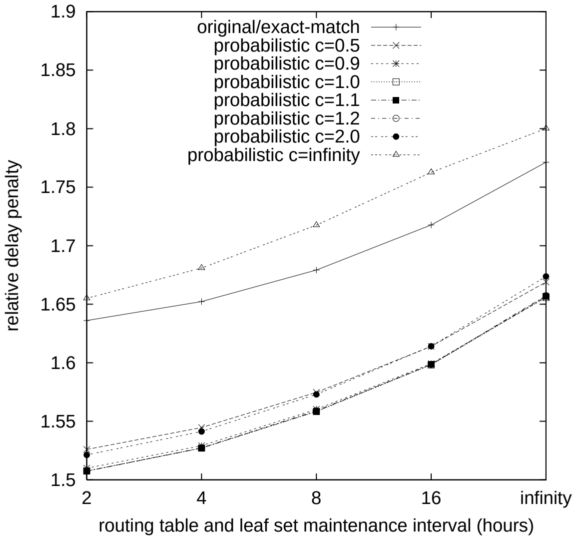

Figure 5. Relative delay penalty when destination is a random key. c =1.0 or 1.1 is optimal in minimizing the delay penalty.

optimization.

Figures 7 and 8 show the routing delay penalty and the average number of routing hops using the probabilistic optimization in the presence of node failures. Since the liveness status of a node in the routing table is updated "lazily", there is a risk involved every time we forward a message to the next hop and this risk becomes higher when the next hop node is chosen by the probabilistic optimization, as follows. The backup nodes are not visited as often as the primary nodes since we don't use them for recovery from failures of primary nodes. Therefore, the possibility that the failures from the backup list remain undiscovered is higher. The overhead caused by failures of the probabilistic routing offsets the potential improvement. We expect that the improvement from the probabilistic routing would dominate if a more proactive mechanism were used to update the liveness information between Pastry nodes and if the backup list liveness states were updated as often as the primary nodes. However, such a more proactive mechanism would also incur additional overheads that offset the benefits.

We also compare the routing performance between the probabilistic optimization and the exact-match optimization when the destination key equals some existing nodeId in the network, since they behave differently in this case. With the exact-match optimization, routing uses the closest candidate in proximity at each hop, and the last hop will use an exact-match in the routing table entry. Under the probabilistic optimization, routing uses the closest candidate in the id space (subject to the threshold criteria), and the last hop will use an exact-match in the routing table entry, same as with the exact-match optimization. One interesting question is

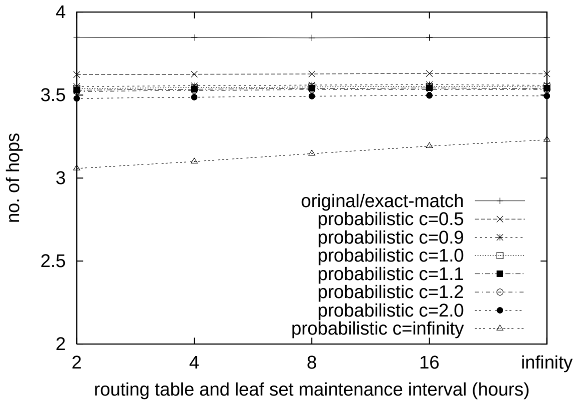

Figure 6. Average number of routing hops when destination is a random key. c =infinite is optimal in minimizing the number of routing hops.

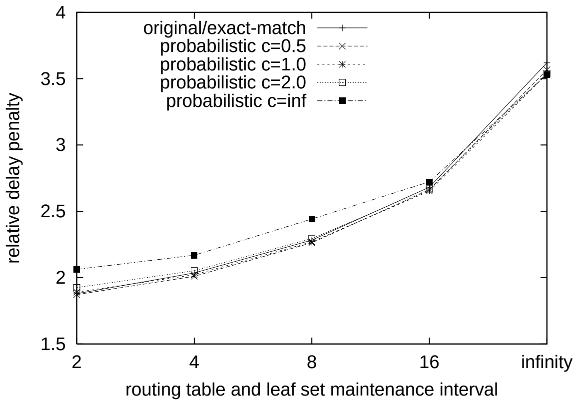

Figure 7. Relative delay penalty when destination is a random key with node failures. c =1.0 is still a good choice.

how the two optimizations compare in this case.

Figures 9-12 show that the two methods have very similar average numbers of routing hops, though increasing the threshold lowers the average number of routing hops for the probabilistic method, with a threshold of infinity minimizing the average number of routing hops. On the other hand, the delay stretch with the probabilistic optimization is higher than with the exact-match optimization. And, the smaller the threshold, the smaller the delay penalty, with the extreme case of a threshold of zero, when the probabilistic method degenerates to the exact-match method. Comparing to optimization 1, this suggests that when the message key is an existing nodeId, the closest node in proximity in the route set is almost as good as any other candidate in terms of approaching the destination in as few hops as possible, but it is a better candidate than others in terms of reducing routing delay.

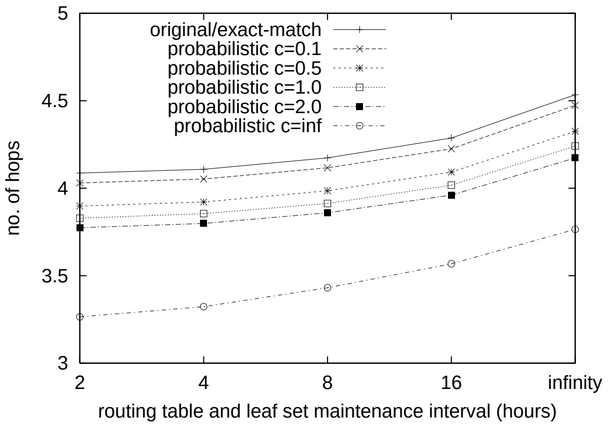

Figure 8. Average number of routing hops when destination is a random key with node failures. $c =$infinite is optimal in minimizing the number of routing hops.

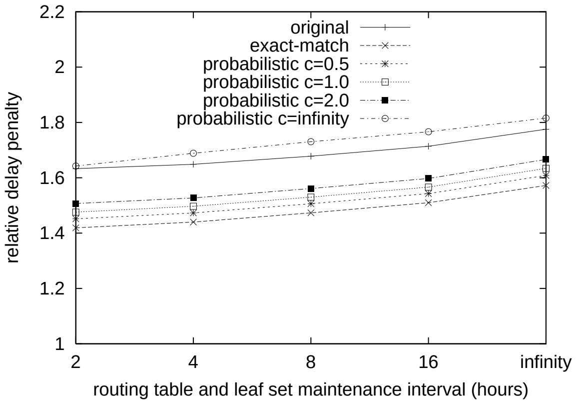

Figure 9. Relative delay penalty when destination is a random existing nodeId. The smaller the c value, the closer the probabilistic method is to the exact method.

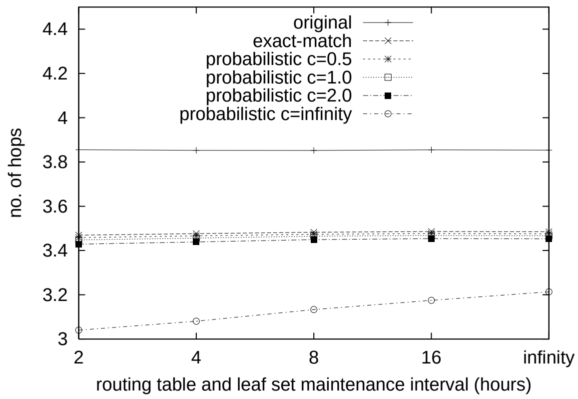

Figure 10. Average number of routing hops when destination is a random existing nodeId. $c =$infinite is optimal in minimizing the number of routing hops.

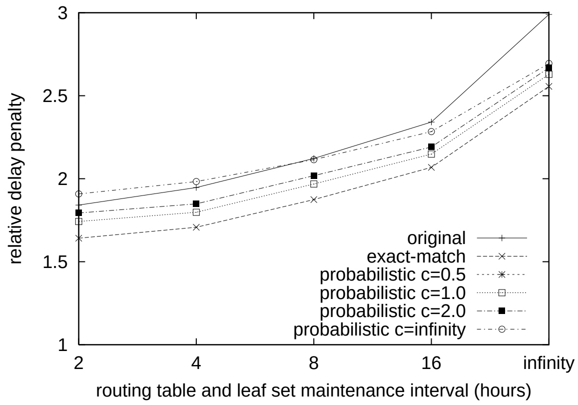

Figure 11. Relative delay penalty when destination is a random existing nodeId with node failures. The smaller the c value, the closer the probabilistic method is to the exact method.

4 Application to other structured p2p overlays

The optimizations studied in this paper are not unique to Pastry. First, all three optimizations can be directly applied to Tapestry [9], which also uses prefix-based routing and proximity neighbor selection. Second, optimization 1 is applicable to Chord. Chord improves its lookup latency via a server selection mechanism as follows: Instead of keeping one node in each entry, k successive nodes can be kept. This is similar to the routing table redundancy in Pastry. Usually these k nodes are equivalent in terms of progress in the id space (although they are different in terms of routing delay), but there is a certain probability that the destination is among them. It is less clear how to apply optimization 2, although intuitively Chord could choose the closest candidate in the proximity space in the early hops (as taking any of the k candidates will make roughly equal progress in the id space), and choose the closest candidate in the id space as the messages approaches the destination.

CAN also has some redundancy mechanisms: multiple realities and overloading coordinate zones. The latter is similar to keeping multiple candidates per routeset in Pastry. It is not obvious how to apply optimizations 1 and 2 presented in this paper to CAN.

5 Related work

The use of "overloading coordinate zones" in CAN is similar to keeping multiple candidates per route set in Pastry. In PAST [4], a peer-to-peer storage utility built on top

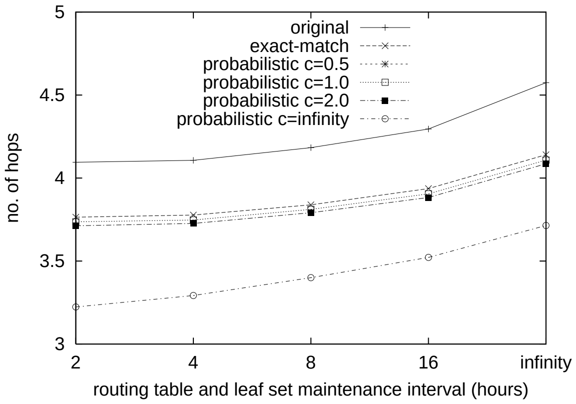

Figure 12. Average number of routing hops when destination is a random existing nodeId with node failures. $c =$infinite is optimal in minimizing the number of routing hops.

of Pastry, each file is inserted with k replicas stored on the k nodes with adjacent nodeIds (by default). Each client lookup request is routed by Pastry towards the k nodes with replicas. To maximize the probability that a client request reaches the nearest of the k replicas (in the proximity space), PAST uses a routing optimization that is similar to optimization 2 (although towards a different goal). At each node along the routing path, PAST looks at the route set and the local leaf set, and estimates if any of the nodes in the leaf set is among the k closest to the key. If so, it forwards the request message to that node.

6 Conclusion

The two optimizations on Pastry routing presented in this paper are of a different nature. Optimization 1 (exact-match) is a simple idea that checks the routing table for direct shortcuts in the id space. It has no downside, but it is only effective when the message key coincides with some existing nodeId. Optimization 2 (probabilistic routing) is essentially a tradeoff between closeness in the id space and closeness in the proximity space, which explains why the optimal value for minimizing the number of hops and the delay penalty are different.

Simulations on a large-scale topology model using Pastry show that both optimizations 1 and 2 yield significant improvements in routing performance. Optimization 1 reduces the average routing delay penalty by about 13% when the message key coincides with some existing nodeId in the network. Optimization 2 with optimal threshold c reduces the average routing delay penalty by 6%–8% for random message keys. We conclude that optimization 2 should be incorporated into Pastry or other p2p overlays if applicable unconditionally. In addition, optimization 1 can be added to Pastry as well, by adding a separate API, to be used when the application has the knowledge that the message key is an existing nodeId, for example, in the TREE messages sent back by the group root in Bayeux [10].

We are evaluating these optimizations using other topology models, such as the Mercator topology and routing models [7]. One interesting question is whether and how much the optimal choice of threshold for the probabilistic method depends on the network topology. We are also interested in studying the impact of these optimizations on the performance of various applications built on top of these p2p overlays, such as distributed storage systems and multicast systems.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *Proceedings of SIGMETRICS*, 2000.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical report MSR-TR-2002-82, 2002.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, August 2001.

[4] A. Rowstron and P. Druschel. PAST: A large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, October 2001.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, August 2001.

[7] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The Impact of Routing Policy on Internet Paths. In *Proceedings of IEEE INFOCOM*, April 2001.

[8] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM*, March 1996.

[9] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-area Location and Routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

[10] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, June 2001.