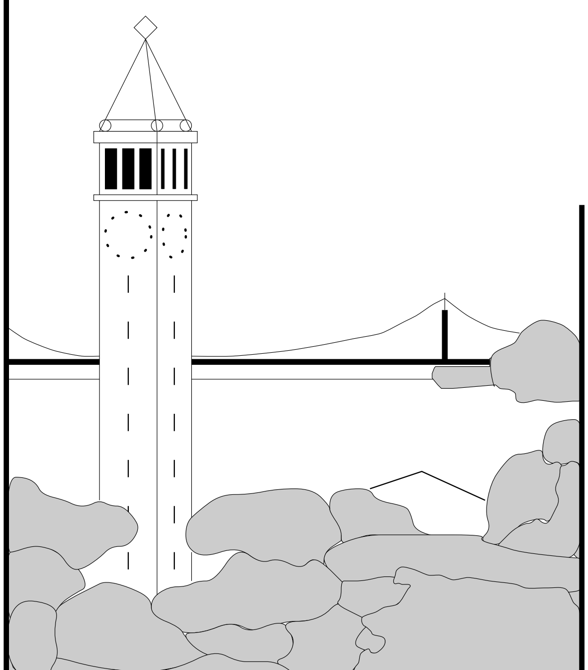
Optimizations for Locality-Aware Structured Peer-to-Peer Overlays

Jeremy Stribling, Kirsten Hildrum, John D. Kubiatowicz

{strib, hildrum, kubitron}@cs.berkeley.edu



Report No. UCB/CSD-03-1266

August 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Optimizations for Locality-Aware Structured Peer-to-Peer Overlays

Jeremy Stribling, Kirsten Hildrum, John D. Kubiatowicz
{strib, hildrum, kubitron}@cs.berkeley.edu

August 2003

Abstract

We present several optimizations aimed at improving the object location performance of locality-aware structured peer-to-peer overlays. We present simulation results that demonstrate the effectiveness of these optimizations in Tapestry, and discuss their usage of the overall storage resources of the system.

1 Introduction

When locating an object using a structured peer-to-peer overlay network, finding the nearest replica of that object is often crucial. Overlays that are locality aware [12] such as Tapestry [11] and Pastry [9] can attempt to locate copies of objects in the local area quickly if they exist, before directing the query down more expensive wide area links. However, even if the object is near the source of a query, it is often the case that one or two hops through the overlay will be needed before the object is found. Since a node with complete routing knowledge (for example, [8]) could have reached this data with a simple direct hop through IP, the extra overlay hops cause a severe relative blowup in the location time of the query, compared to the minimum possible.

This inefficiency can be measured quantitatively by the *Relative Delay Penalty* (RDP) of the query, also known as *stretch*. For this paper, we define RDP as the ratio of the distance a query travels through the overlay network to an object and the minimal distance to that object (i.e. through IP). When the data is located outside the local area of the querying node, the RDP for an overlay like Tapestry has been shown experimentally to be small, but when the data is nearby the RDP can be very large [11]. For applications that depend on finding nearby replicas quickly for reasonable performance, such as web caching, this extra latency overhead may be unacceptable.

While previous work has explored the importance of locality in overlay performance [1, 2, 6, 7, 12], in this paper we focus on various optimizations for the object publication and location algorithms of locality-aware overlays. Overlays that can benefit from these optimizations must support the Decentralized Object Location and Routing (DOLR) interface [3], allowing clients to locate and route to an object knowing only the object's location-independent ID. The network proximity [2] aspects of the DOLR routing mechanism provide the locality-awareness necessary to find the nearest copy of an object. We will examine ways to make object location in such an overlay very efficient, given a layer of object pointer indirection (i.e. the objects stored in the DOLR are pointers to the location of the actual data). Specifically, we examine the tradeoff between object pointer state and local area RDP improvement in Tapestry, and show that a large reduction in RDP is possible by adding relatively few additional object pointers into the system. After a brief description of the Tapestry publication and location algorithms in Section 2, Section 3 will outline several optimization strategies. We will present a quantitative evaluation of these optimizations in Section 4, and conclude in Section 5.

2 Basic Tapestry Algorithms

In Tapestry, when a node wishes to advertise the fact that it has a certain object available, it sends a publish message into the overlay toward the object's unique root node.[1] At each overlay hop, including the root itself, a pointer to the location of the object is saved. When another node wants to locate this object, it sends a location message toward the root node of the object. Each node along the path of this message checks whether it has a pointer to the object. If so, the location query gets forwarded directly to the node that published the object; otherwise, the query continues along

This research supported by NSF career award #ANI-9985250, NFS ITR award #CCR-0085899, and California MICRO award #00-049.

[1] The relationship between an object and its root node is established deterministically and dynamically as a function of the object's unique ID and the IDs of all nodes currently in the network.

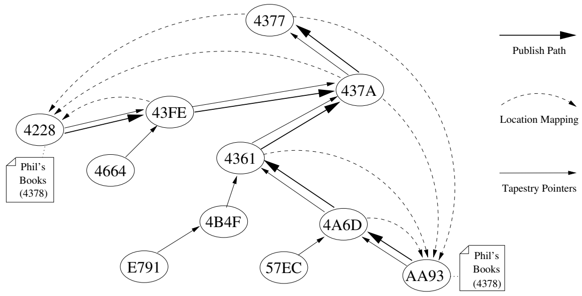


Figure 1: *Tapestry object publish example.* Two copies of an object (`4378`) are published to their root node at `4377`. Publish messages are routed to root, depositing a location pointer for the object at each hop encountered along the way. Only relevant Tapestry pointers are shown.

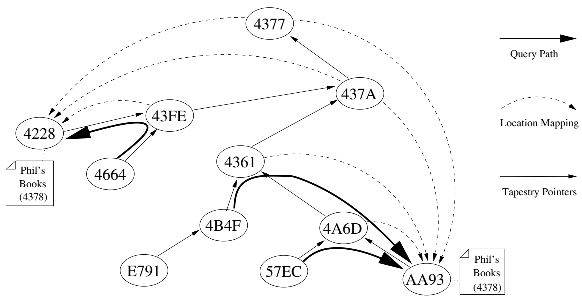


Figure 2: *Tapestry route to object example.* Several nodes send messages to object `4378` from different points in the network. The messages route towards the root node of `4378`. When they intersect the publish path, they follow the location pointer to the nearest copy of the object.

the path to the object's root node, where a pointer is guaranteed to exist. When objects are replicated and multiple nodes publish the same object, each overlay node keeps a list of all pointers it has received for each object, sorted by the network distance from that node to the publisher. It will direct queries for that object to the nearest publisher. Figures 1 and 2 depict this process; see [11] for further discussion of these algorithms and an explanation for how nearby object replicas are likely to be found before ones that are further away. As we discussed in Section 1, however, any extra latency at all has a large effect on RDP when finding a local area object since the object is already so close. Next we describe a few techniques for lowering the RDP in such situations.

3 Optimizations

In this section, we describe in detail three different optimizations that improve local area object location RDP: publishing to backups, publishing to nearest neighbors, and publishing to the local surrogate.

3.1 Publishing to Backups

When forming its routing table, a Tapestry node usually has a choice between several nodes for each entry. [5] explains how Tapestry's tree-like structure makes it flexible in that respect. It chooses the closest of these nodes to be the *primary neighbor*, which is the node that will serve as the next hop for messages heading through that entry of the table. For purposes of fault tolerance, however, each entry can hold up to c neighbors; thus, up to $c - 1$ nodes not chosen to be the primary neighbor become *backup neighbors* in the entry, and are sorted within the entry by network distance. For some entries there may be many nodes from which to choose, and it is likely that some of the backup neighbors chosen will be nearly as close as the primary, given that the overlay is sufficiently dense in the area. These backup neighbors are also likely to be primary neighbors in the routing tables of other nodes in the local area.

Our first optimization takes advantage of these properties during the object publication process to deposit more object pointers in the local area around the publisher. At each hop along the publication path, the current node forwards the publish message to as many as b backup neighbors, in addition to forwarding it to the primary neighbor. To bound the state consumed by these additional pointers and because only the first few nodes on the path are likely to be in the object's local area, this optimization only occurs along the first h hops of the publication path. Nodes that receive

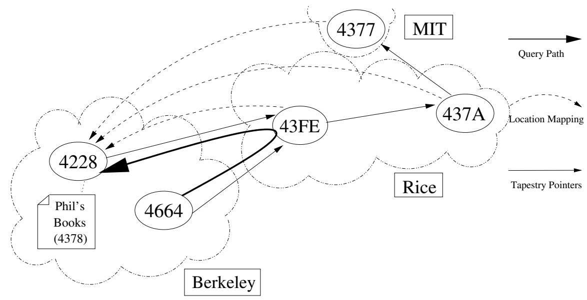


Figure 3: *Route to object example, with local areas shown.* A possible grouping of nodes from Figure 2 into local areas.

pointers due to this optimization do not forward the publish messages.

Since these backup neighbors are probably in the local area of the publisher, and will serve as primary neighbors to nearby nodes that may be locating the object, the location query paths starting from nodes in the local area are likely to encounter these additional pointers. Take for example the network illustrated in Figures 1 and 2, where we assume that node `4B4F` is a backup neighbor for node `4A6D` in node `AA93`'s routing table. If this optimization is applied during the publication of object `4378`, with a value of one for both b and h, node `4B4F` will receive an additional pointer for the object. Then, when node `4B4F` begins its query, it will be able to jump directly to the object, reducing the RDP for the query to one.

3.2 Publishing to Nearest Neighbors

Our second technique for placing additional object pointers in the local area is a form of limited pointer flooding. Rather than restricting the additional pointers to only backup neighbors as in the previous section, nodes forward publish messages to any nearby neighbor at a given level. At each hop along the path, the algorithm finds the closest n nodes at the same routing table level as the next hop, and places additional pointers on those nodes. Again, this is bounded to the first h hops of the path, and those nodes do not forward the publish message any further.

Note that if n is large enough, this technique effectively floods the local area with object pointers, so that almost every nearby node that queries for the object will already have a pointer to that object, similiar to global knowledge. This is of course very good from an RDP standpoint, but could be very costly in terms of storage. This tradeoff is explored quantitatively in Section 4.

3.3 Publishing to the Local Surrogate

A more complicated optimization is based on the observation that wide area hops are likely to be many times longer than local area hops. Thus, if an object is in the local area, but a query happens to venture into the wide area before finding a pointer to the object, the effect on RDP will be disastrous. Figure 3 illustrates such a scenario. To avoid this unfortunate situation, a pointer can be placed on an object's *local surrogate* (the node in the local area that would serve as the object's root, if no nodes outside the local area existed) during publication. Then the local surrogate can be checked before leaving the local area during an object location query, hopefully avoiding a costly and possibly unnecessary wide area hop. Note that this technique occurs naturally in some systems [1, 4, 6]. If applied to the situation in Figure 3, a pointer to object `4378` would be placed on its local surrogate node `4664`, allowing `4664` to find the object directly during its query without leaving the local area. We do not bound this optimization by number of hops, because it is already very restrained about where it places additional pointers.

An obvious issue with this scheme is deciding when the next hop will take the query out of the local area. One simple heuristic is to classify the next hop as a wide area hop if the latency of that link is more than t times the latency of the last link traveled (where t is a given threshold value). A more sophisticated technique could use learning strategies to adjust t automatically and dynamically based on the current characteristics of the network.

4 Results

To demonstrate the effectiveness of the optimizations we enumerated in Section 3, and to examine the tradeoff between storage overhead and local area RDP, we ran several tests in simulation. Here we describe our experimental setup, and follow that with a presentation and analysis of the results from our experiments.

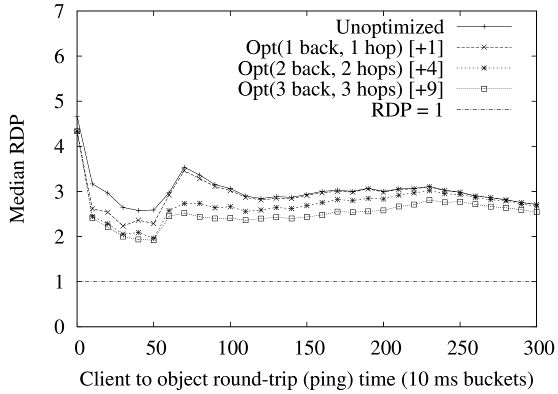


Figure 4: *The effect of publishing to backups on median RDP.* Shows the median RDP for object location using b backups and h hops with analytical cost (additional pointers per object) shown in brackets.

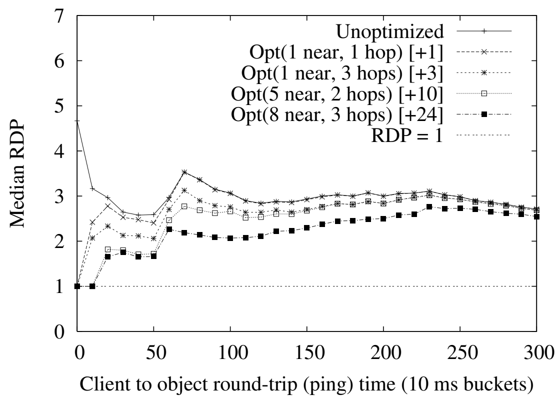


Figure 5: *The effect of publishing to nearest neighbors on median RDP.* Shows the median RDP for object location using n neighbors and h hops with analytical cost (additional pointers per object) shown in brackets.

4.1 Experimental Setup

We implemented these optimizations in the current Berkeley version of Tapestry, which is written in Java. In order to perform large-scale and repeatable experiments, we used the simulator first described in [7] to provide an event-driven network layer that simulates network delays, based on a GT-ITM transit stub model [10]. The transit stub graph we used for these experiments consists of 1092 nodes, with approximately 30 nodes per stub domain. Out of these physical nodes, 1090 participate in the Tapestry network to demonstrate the effect of the optimizations on dense networks. The Tapestry nodes use 40-digit IDs of base 4, and the number of nodes per routing table entry, c, is 4. Due to inflated network distances assigned automatically by the graph-generating program, in the graphs we present the network distance between nodes has been linearly scaled to reflect a real network more accurately.

In these experiments, each Tapestry node publishes 25 objects with random IDs. Each node then locates 100 objects, chosen randomly from the set of all published objects, and calculates the RDP for each query.

4.2 Median RDP Improvements

Figure 4 shows how different values of b in the publishing to backups optimization (see Section 3.1) affect median object location RDP. The graph shows that, as expected, the optimization is most effective when the query source is relatively close to the object, lowering the median RDP by as much as one point in some places. Note that as the objects get farther away, the optimized lines converge to the unoptimized line. In this case, the decrease in RDP is limited by the number of backups in each route entry.

The publishing to nearest neighbors algorithm (see Section 3.2), by contrast, is limited by the number of neighbors on a certain level of the routing table, and thus a larger improvement is possible. Figure 5 shows that for very nearby objects, the median RDP can be lowered to a minimum, if one is willing to allow a large storage overhead per object.

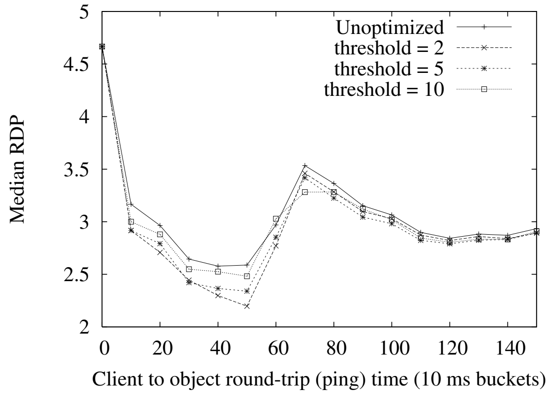


Figure 6: *The effect of publishing to the local surrogate on median RDP.* Shows the median RDP for object location using threshold t. Note the scale of this graph differs to show greater detail.

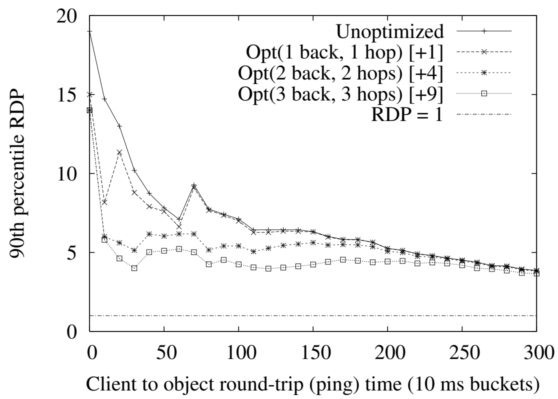


Figure 7: *The effect of publishing to backups on* 90^{th} *percentile RDP.*

Figure 6 illustrates the relationship between the RDP and object distance when publishing to the local surrogate (see Section 3.3) for various values of t. By adjusting t, we can subtly affect the object location performance in the local area, which bodes well for more sophisticated algorithms that adjust t automatically.

4.3 90^{th} percentile RDP Improvements

Reducing the variance of the RDP is important for ensuring that the majority of queries complete efficiently. High variance indicates client/server pairs that will consistently see non-ideal performance and tends to limit the advantages that clients gain through careful object placement. We measure this variance with the 90^{th} percentile of the RDP; if 90% of queries perform efficiently, we can be confident that our optimizations are aiding the majority of client/server pairs. Figures 7, 8 and 9 illustrate the improvement in 90^{th} percentile RDP, demonstrating the effectiveness of our optimizations at reducing variance.

In particular, notice that the local surrogate optimization gives a rather large savings in 90^{th} percentile RDP (Figure 9) when compared to its median improvement (Figure 6). [2] In some places it does better than the other optimizations, and at much lower cost (see Section 4.4). For each optimization, in fact, we observe a substantial savings in 90^{th} percentile RDP (almost nineteen points in Figure 8), clearly showing that the optimizations improve nearly all inefficient cases of local area object location.

4.4 Cost

These improvements in local area RDP come at the cost of increased storage overhead; there will be many more pointers per object in the system with optimizations than without. Table 1 compares the calculated cost of an optimization

[2] Note that the scales of these two graphs differ.

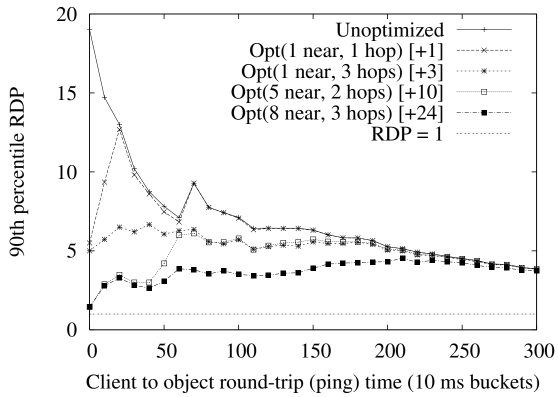


Figure 8: *The effect of publishing to nearest neighbors on 90^{th} percentile RDP.*

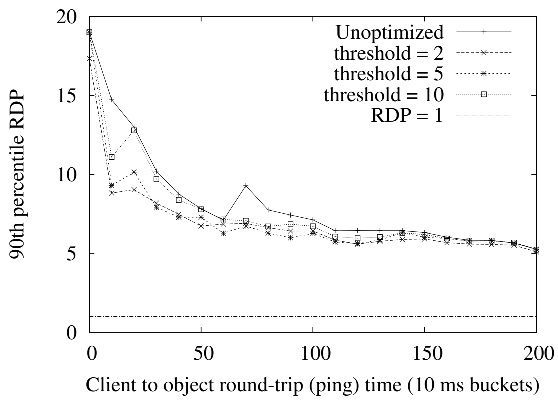


Figure 9: *The effect of publishing to the local surrogate on 90^{th} percentile RDP.*

(the maximum number of additional pointers per object possible) to the observed cost (the average number of additional pointers per object stored during the experiment), for various optimization parameters. Note that the observed cost is always less than the analytical cost (sometimes substantially less). This is because many times a routing table entry or level is not entirely full, and thus it is not possible (or necessary) to publish the full number of additional pointers.

In the case of publishing to the local surrogate, the cost depends entirely on the network characteristics, and thus precomputing an analytical cost is not possible. Although this optimization gives slighter gains in RDP than the other two, it is more intelligent about where it places the additional object pointers, using storage resources very efficiently. Our future work for this optimization includes experimenting with more sophisticated techniques for identifying wide area links, hopefully achieving an even lower RDP for the same efficient resource usage.

4.5 Combined Optimizations

In an effort to examine how the optimizations interact with each other, we ran simulations with different parameter combinations for publishing to b backups and n nearest neighbors. Figures 10 and 11 show the results for median RDP and 90^{th} percentile RDP, respectively. Both of these optimizations overshadow the subtle effects of the local surrogate optimization when combined with it, and thus we delay the presentation of these comparisons until we develop more sophisticated techniques for determining the local surrogate.

Because there is much more freedom during publication in choosing nearest neighbors than in choosing backups, the nearest neighbors optimization clearly influences the behavior of the combined optimizations more greatly. However, the are subtle differences. For example, we include on the graphs two experiments with analytical cost ten, one placing additional pointers on nearest neighbors only (b = 0, n = 5, h = 2), and another combining the two optimizations (b = 2, n = 3, h = 2). In the local area these perform similarly, but the combined optimizations outperform

b	h	Analytical cost	Observed cost
1	1	1	.83
2	2	4	3.11
3	3	9	6.18

a) Publishing to backups

n	h	Analytical cost	Observed cost
1	1	1	.99
1	3	3	2.75
5	2	10	9.12
8	3	24	16.77

b) Publishing to nearest neighbors

t	Observed cost
2	.38
5	.24
10	.11

c) Publishing to the local surrogate

Table 1: *Cost of optimizations.* A summary of the observed cost of an optimization versus the analytical (maximum) cost. Costs are given in average additional pointers per object in the network. Observed costs tend to be much lower than analytical costs due to unfilled routing table entries and levels.

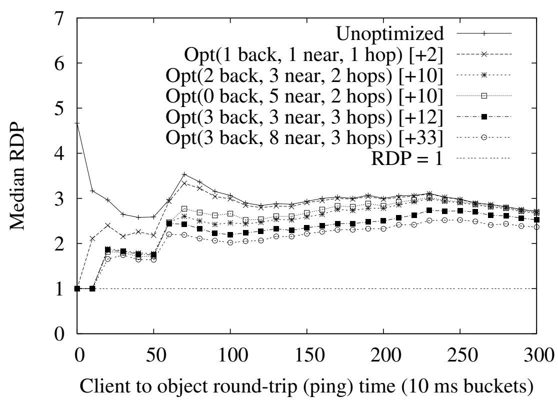


Figure 10: *The effect of publishing to backups and nearest neighbors on median RDP.*

the single nearest neighbor optimization for objects not in the local area (specifically, note the differences for objects between 50 and 150 ms away). Moreover, Table 2 indicates that though they have the same analytical cost, in practice the case of combined optimizations is actually less expensive. This demonstrates that the brute force method of local area flooding used in the nearest neighbor optimization can be combined with the careful but limited placement used in the backups optimization in ways that lead to more intelligent and efficient pointer placement.

5 Conclusion

We have discussed three different optimizations for locality-aware structured peer-to-peer overlays, and shown their effectiveness at reducing the RDP of locating nearby objects. We found that by spending storage to house additional object pointers in the system, local area RDP can be greatly improved; furthermore, if the optimization technique is conservative and judicious about where it places the additional pointers, a very small storage overhead can result in a respectable savings in RDP. Although in this paper we have focused on the implementation and effects of these techniques in Tapestry, we believe they can be applied to other DOLRs as well, such as Pastry (given a pointer indirection layer). From the results of this paper, we contend that further research into this subject would be useful to applications that depend on locality-aware overlays for their performance, and our future work will focus on testing the impact of these optimizations on actual applications, as well as on evaluating the cost to the DOLR of maintaining the extra pointers.

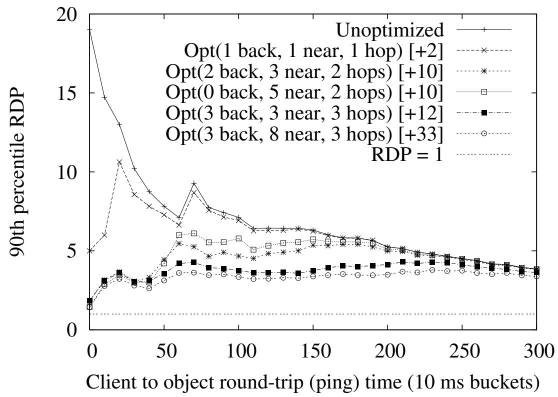


Figure 11: *The effect of publishing to backups and nearest neighbors on 90^{th} percentile RDP.*

b	n	h	Analytical cost	Observed cost
1	1	1	2	1.77
2	3	2	10	7.37
0	5	2	10	9.12
3	3	3	12	10.61
3	8	4	33	20.00

Table 2: *Cost of combined optimizations.*

Acknowledgments

We thank Ben Zhao, Anthony Joseph, and Sean Rhea for their help in developing these optimizations, and for comments on earlier drafts of this paper.

References

[1] ABRAHAM, I., MALKHI, D., AND DOBZINSKI, O. LAND: Locality aware networks for distributed hash tables. Tech. Rep. TR 2003-75, Leibnitz Center, The Hebrew University, June 2003.

[2] CASTRO, M., DRUSCHEL, P., HU, Y. C., AND ROWSTRON, A. Exploiting network proximity in peer-to-peer overlay networks. Tech. Rep. MSR-TR-2002-82, Microsoft Research, 2002.

[3] DABEK, F., ZHAO, B., DRUSCHEL, P., KUBIATOWICZ, J., AND STOICA, I. Towards a common API for structured P2P overlays. In *Proc. of IPTPS* (February 2003).

[4] FREEDMAN, M. J., AND MAZIERES, D. Sloppy hashing and self-organizing clusters. In *Proc. of IPTPS* (February 2003).

[5] GUMMADI, K., GUMMADI, R., GRIBBLE, S., RATNASAMY, S., SHENKER, S., AND STOICA, I. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of SIGCOMM* (August 2003), ACM.

[6] HARVEY, N. J., JONES, M. B., SAROIU, S., THEIMER, M., AND WOLMAN, A. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of USITS* (March 2003).

[7] RHEA, S., AND KUBIATOWICZ, J. Probabilistic location and routing. In *Proc. of INFOCOM* (June 2002), IEEE.

[8] RODRIGUES, R., LISKOV, B., AND SHRIRA, L. The design of a robust peer-to-peer system. In *Proc. of SIGOPS European Workshop* (September 2002).

[9] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware* (November 2001).

[10] ZEGURA, E., CALVERT, K., AND BHATTACHARJEE, S. How to model an internetwork. In *Proc. of INFOCOM* (1996).

[11] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications* (November 2003). Special Issue on Service Overlay Networks, to appear.

[12] ZHAO, B. Y., JOSEPH, A. D., AND KUBIATOWICZ, J. Locality-aware mechanisms for large-scale networks. In *Proc. of International Workshop on Future Directions of Distributed Systems* (June 2002).