Exploring RDP in Locality-Aware Structured Peer-to-Peer Overlays

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

Abstract

We present new insight into the object location performance of locality-aware structured peer-to-peer overlays, as measured by relative delay penalty (RDP), and find that the problem of high RDP between nearby nodes can be explained theoretically. We discuss a simple optimization aimed at reducing the RDP between nearby nodes by strategically placing extra object pointers, and evaluate its effectiveness via simulation of a Tapestry network. We find that for a modest pointer overhead factor of 3.11, we can improve mean local area RDP by 38%, and 90^{th} percentile local area RDP by 50%.

1 Introduction

When locating an object using a structured peer-to-peer overlay network, finding the nearest replica of that object is often beneficial. Overlays that are locality aware [13] such as Tapestry [12] and Pastry [8] 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 generally the case that a few hops through the overlay will be needed before the object is found. Since a node with fully-distributed routing knowledge (for example, [7]) 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 *location 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 [12]. For applications that depend on finding nearby replicas quickly for reasonable performance, such as web caching, this extra overhead may be unacceptable.

While previous work has explored the importance of locality in overlay performance [1–4, 6, 13], we present new theoretical insights explaining *why* we expect RDP to be high between nearby nodes, and show that simply having a locality-aware overlay isn't enough to eliminate RDP inefficiencies. We then present one optimization for the publication algorithm of such overlays, designed to help reduce location RDP in these situations. Unlike previous schemes that minimize *overall* RDP (such as the PNS/PRS schemes detailed in [3]), we focus on minimizing the *local area* RDP. Furthermore, we aim to improve performance at a stronger level than a metric like local convergence [2, 3]: we want paths between query sources and publishers to converge as quickly as possible even within the local area, rather than simply converging before leaving the local area. 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.

In the next section, we describe our new theoretical framework for understanding location RDP at small distances, and explain why such RDP matters.

2 Location RDP at Small Distances

Intuitively, we expect location RDP to be higher at small distances than at longer distances. To see this, consider the following thought experiment.

Place n overlay nodes at the integers on the number line, from 1 to n, such that adjacent nodes are separated by a distance of one. For simplicity, we will refer to each node by its position, and we assume that n is a power of 2. A particular object ID and the routes taken to that ID through the overlay create a logical tree on these nodes. Specifically, in a base-2 Tapestry tree, roughly one-half of the nodes are in the bottom level in the tree, one-quarter are in the next level, etc. Suppose the tree is "perfect", meaning that exactly every other node is a leaf, and consider the pair of nodes $(2k-1, 2k)$, for some $k > 0$.

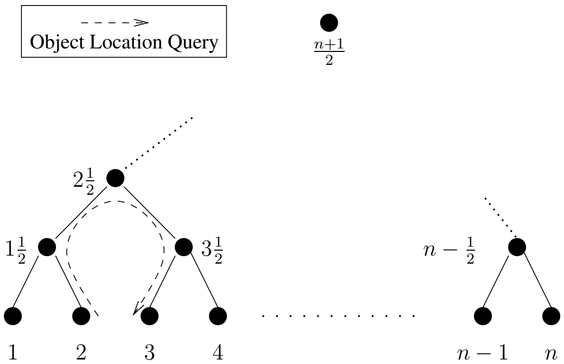

Figure 1: *A perfect base-2 Tapestry tree, with hypothetical average parents.* A sample object location query with an RDP of three is shown.

If $2k$ is a leaf, then $2k$'s distance to its parent $2k-1$ is one, and $2k-1$'s distance to its parent (itself) is zero; the average distance from a node to its parent is $\frac{1}{2}$. To simplify the calculation, imagine a hypothetical average parent located at a distance of $\frac{1}{2}$ from both nodes, at $2k-\frac{1}{2}$. These are not physical nodes; rather, they represent the average length of the first hop of a route. We repeat this process at each level of the logical routing tree, and get the picture in Figure 1.

In Tapestry, objects are *published* by placing pointers to them at each node along the path from the publisher to the root, and object location proceeds by checking for pointers along the path from the query source to the root. The RDP is determined by where the publish and search path first intersect, or in other words, the least common ancestor of the publisher and the query source.

2.1 Analytical RDP

We now calculate the average location RDP for nearby nodes. For us, "nearby" means that the publisher and the searcher are distance one apart. Half such pairs of nearby nodes share the same parent, and so have location RDP of one, since the request need only travel a distance of one to reach the publisher. One quarter of the pairs share the same grandparent, but not the same parent; for these nodes, the RDP is three (see Figure 1 for an example). In general, if $i = \log_2 n$, for any positive $j \leq i$ there are 2^{i-j} pairs with location RDP $2^j - 1$. The average RDP is then $\frac{1}{n}\sum_{j=1}^{i} 2^{i-j}(2^j-1)$, or $i - \sum_{j=1}^{i}\frac{1}{2^j}$, which is less than or equal to $\log n - 1$. Thus, the average RDP is $O(\log n)$. A similar argument shows that for pairs at distance f, the average RDP is $O(\lceil \log(n/f) \rceil)$. [1]

[1] Note that since $f \leq n-1$, $\lceil \log(n/f) \rceil \geq 1$.

The intuition behind the math is that a given point k needs to be able to route quickly to both $k+1$ and $k-1$, and a binary tree only allows it to share a parent with one or the other, forcing it to pay more to reach the other nearby node.

This is only intended as an informal argument, but the phenomenon discussed here is quite general and could be formalized using techniques similar to those in [5]. In particular, note the following.

- This is only worse in more general metric spaces. Consider, for example, a d-dimensional grid. In a line, the point k is close to two other points. In a d-dimensional grid, it is close to $O(2^d)$ points. In a transit-stub topology, k will be close to all nodes in its stub and far from all others, exacerbating the gap between wide area RDP and local area RDP.
- Increasing the degree of the tree does not solve this problem. Though it reduces the number of nodes with neighbors not sharing a parent, it increases the average distance to the parent.
- While we framed this argument around Tapestry, its implications are much broader. It suggests that a tree cannot provide constant location RDP for nearby objects. This is significant since in most structured peer-to-peer overlays (even those using Proximity Route Selection [3]), the set of lookup paths from different sources to the same ID forms a tree. Any tree-based set of lookup paths is subject to the difficulty described here, even when using caching along the intermediate path.

Finally, notice that the RDP for pairs that are far apart (say, separated by more than $\frac{n}{2}$) is quite low in this example. Since most pairs are far apart (one-quarter of all pairs are at distance more than $\frac{n}{2}$), the overall average RDP is low. This suggests that average overall RDP (and median overall RDP, as explored in [3]) is an incomplete measurement, since it is not really affected by the important cases of close objects (see Section 2.3).

2.2 Simulated RDP

To validate the above thought experiment, we present the RDP distribution of individual object location queries between nearby nodes, from a Tapestry simulation on a transit-stub topology, in Figure 2. While the exact experimental setup is detailed in Section 4.1, at a high level we can see from this histogram that although the location RDP of most

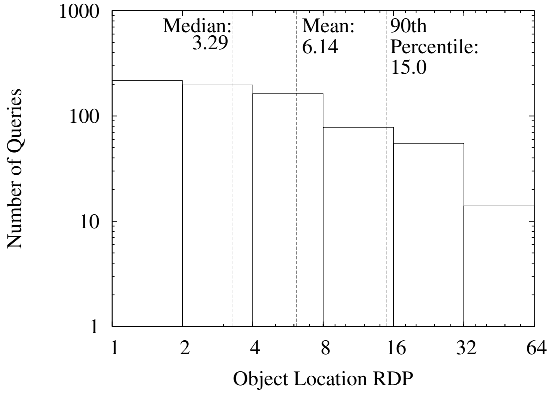

Figure 2: *RDP in a simulated transit-stub topology, unoptimized.* Distribution of location RDP in buckets of exponentially-increasing size, for pairs of nodes with ping times no bigger than 25 ms. The height of each bucket represents the number of queries between such nodes with an RDP of corresponding magnitude.

queries between nearby nodes is small, for some queries it is very large. In fact, this is what our thought experiment predicts: since the RDP experienced by a pair of nodes depends on their least common ancestor in the tree, we would expect the number of pairs with a least common ancestor at level k to decrease exponentially in k, while the RDP experienced should increase exponentially. Note also that there is nearly a factor of two difference between the mean and median (and more than a factor of four between the median and the 90^{th} percentile), suggesting that the local area RDP measurements experienced by lookups can vary widely. The overall mean RDP for queries between any two nodes is just 3.01 (less than half of the mean RDP for close objects), demonstrating that indeed overall measurements can obscure information about the RDP between nearby nodes.

2.3 Importance of RDP Reduction

Large RDPs between nearby nodes, such as those shown in Figure 2, often indicate an object location query path that leaves the local stub unnecessarily. Such paths not only present performance problems revealed by RDP measurements, but will also consume limited, expensive wide area bandwidth, and the traffic will be more likely to suffer from router queuing effects and dropped packets. Thus, keeping a query local to a stub if possible can greatly increase overall performance. [10] discusses an optimization designed specifically to avoid unnecessary interstub communication.

Furthermore, consider the case in which nodes publish a particular ID identifying themselves as a service provider or part of a multicast group, as in $i3$ [9]. Every message routed to that ID must go through the overlay (as opposed to the case of object location, in which a direct download connection can be instantiated once the location of the object is known). Hence, any latency overhead added to the path affects every message routed to that ID, and may drastically increase the cost of using nearby services or multicasting to nearby nodes.

The following section describes an optimization that acheives the goal of RDP reduction by adding extra pointers to the system.

3 Optimization: Publishing To Backups

In this section, we describe an optimization that improves local area location RDP. Although we present the optimization in the context of Tapestry, we maintain it would be effective in any overlay network that allows clients to locate and route to an object knowing only the object's location-independent ID, and uses network proximity [2] techniques to find the nearest copy of an object. Our optimization makes object location in such an overlay more efficient, given a layer of object pointer indirection (i.e. the objects stored in the network are pointers to the location of the actual data). See [10] for the discussion and analysis of further optimizations.

When forming its routing table, a Tapestry node usually has a choice between several nodes for each entry. It chooses neighbors from the possible set of nodes using Proximity Neighbor Selection (PNS) [3]; the closest of these nodes becomes the *primary neighbor*, which is the node that will serve as the next hop for identifiers mapping to 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 by latency within the entry. 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. These backup neighbors are also likely to be primary neighbors in the routing tables of other local area nodes.

We take 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 pointers due to this optimization do not forward them further.

Recall that in Section 2 we argued that the RDP was high between nearby nodes because a point k was physically close to many points but, in a binary tree, could only associate itself with exactly one. Adding additional pointers on backups as described above is essentially allowing the point k to associate itself with multiple parents in the tree. These backups can be used either on search (that is, publish as normal, but search using the backups), or on publish (search as normal, but publish to the backups). Since searching is expected to be more frequent than publishing, we use them on publish to save bandwidth.

4 Results

To demonstrate the effectiveness of this optimization, 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.

4.1 Experimental Setup

We implemented this optimization 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 [6] to provide an event-driven network layer that simulates network delays, based on a GT-ITM transit stub model [11]. 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; they build their routing tables using ideal PNS. Due to inflated network distances assigned automatically by the graph-generating program, the network distance between nodes was linearly scaled to reflect real networks more accurately. [2] In these experiments, each Tapestry node publishes 25 objects with random IDs. Each node then locates 100 randomly-chosen objects.

[2] We also ran these experiments on the GT-ITM latency distribution described in [3], and the results were qualitatively similar. We omit these graphs due to space limitations.

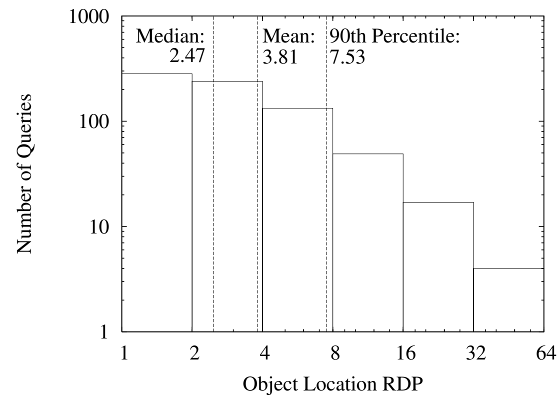

Figure 3: *RDP in a simulated transit-stub topology, publishing to backups.* Distribution of location RDP using our optimization with b = 2 and h = 2, for pairs of nodes with ping times no bigger than 25 ms.

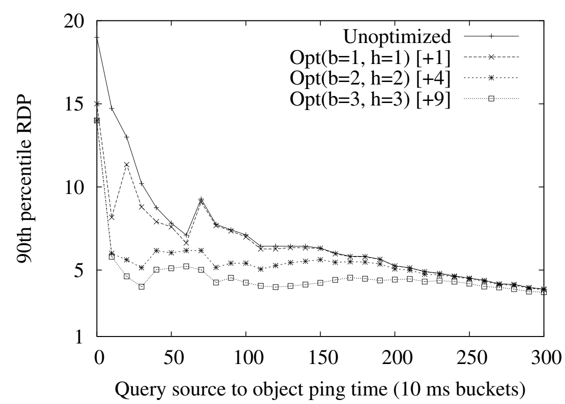

Figure 4: *The effect of publishing to backups on* 90^{th} *percentile RDP.* Results of our optimization using b backups and h hops with analytical cost (additional pointers per object) shown in brackets, for various query-source-to-object distances.

4.2 RDP Distribution Improvements

Figure 3 shows the effect of using two backups on the RDP distribution of queries between nearby nodes. Note in particular that this optimization greatly reduces the mean RDP when compared with Figure 2, as well as bringing it closer to the median, and reduces the 90^{th} percentile RDP by half.

4.3 90^{th} percentile RDP Improvements

We would also like to evaluate the effectiveness of our optimization as a function of the distance between any pair of nodes. Furthermore, 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

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

Table 1: *Cost of publishing to backups.* A summary of the observed cost of our optimization versus the analytical (maximum) cost, in average additional pointers per object. Observed costs tend to be much lower than analytical costs due to unfilled routing table entries and levels.

the advantages that clients gain through careful object placement. To this end, Figure 4 shows how different values of b and h affect 90^{th} percentile location RDP between pairs of nodes at varying distances. The graph shows that, as expected, the optimization is most effective when the query source is relatively close to the object, lowering the 90^{th} percentile RDP by nearly ten points in some places. This indicates that the variance of the RDP between nearby nodes is similarly reduced. Also note that as the objects get farther away, the optimized lines converge to the unoptimized line.

4.4 Cost

These improvements in local area RDP come at the cost of increased storage overhead; there will be more pointers per object in the system with optimizations than without. Table 1 compares the calculated cost of the optimization (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 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 extra pointers. In [10] we present other optimizations with even better cost-to-improvement ratios.

5 Conclusion

We have discussed the problem of high RDP between nearby nodes in a locality-aware overlay network, and shown a new way to understand the issue theoretically. Next, we presented an optimization for such overlays, and demonstrated its 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. We believe that both RDP and local convergence [2, 3] are important metrics of the performance of peer-to-peer overlays, and a subject of future work is to determine by how much our optimization improves local convergence. Other future work will focus on simulating these experiments on more realistic topologies, testing the impact of these optimizations on actual applications, and evaluating the cost of maintaining the extra pointers.

References

[1] ABRAHAM, I., MALKHI, D., AND DOBZINSKI, O. LAND: Stretch (1+ϵ) locality-aware networks for DHTs. In *Proc. of ACM SODA* (January 2004).

[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] 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.

[4] 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).

[5] KRAUTHGAMER, R., AND LEE, J. R. Navigating nets: Simple algorithms for proximity search. In *Proc. of ACM SODA* (January 2004).

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

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

[8] 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).

[9] STOICA, I., ADKINS, D., RATNASAMY, S., SHENKER, S., SURANA, S., AND ZHUANG, S. Internet indirection infrastructure. In *Proc. of IPTPS* (March 2002).

[10] STRIBLING, J., HILDRUM, K., AND KUBIATOWICZ, J. D. Optimizations for locality-aware structured peer-to-peer overlays. Tech. Rep. UCB//CSD-03-1266, U. C. Berkeley, August 2003.

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

[12] 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*. Special Issue on Service Overlay Networks, to appear.

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