Exploiting Network Proximity in Distributed Hash Tables

Miguel Castro[1] Peter Druschel[2] Y. Charlie Hu[3] Antony Rowstron[1]

[1]Microsoft Research, 7 J J Thomson Close, Cambridge, CB3 0FB, UK.
[2]Rice University, 6100 Main Street, MS-132, Houston, TX 77005, USA.
[3]Purdue University, 1285 EE Building, West Lafayette, IN 47907, USA.

Abstract

Self-organizing peer-to-peer (p2p) overlay networks like CAN, Chord, Pastry and Tapestry (also called distributed hash tables or DHTs) offer a novel platform for a variety of scalable and decentralized distributed applications. These systems provide efficient and fault-tolerant routing, object location, and load balancing within a self-organizing overlay network. One important aspect of these systems is how they exploit network proximity in the underlying Internet.

Three basic approaches have been proposed to exploit network proximity in DHTs, geographic layout, proximity routing *and* proximity neighbour selection. *In this position paper, we briefly discuss the three approaches, contrast their strengths and shortcomings, and consider their applicability in the different DHT routing protocols. We conclude that proximity neighbor selection, when used in DHTs with prefix-based routing like Pastry and Tapestry, is highly effective and appears to dominate the other approaches.*

Introduction

Several recent systems (CAN [4], Chord [9], Pastry [6] and Tapestry [10]) provide a self-organizing substrate for large-scale peer-to-peer applications. These systems can be viewed as providing a scalable, fault-tolerant distributed hash table (DHT), in which any item can be located within a bounded number of routing hops, using a small per-node routing table.

DHTs assign a live node in the overlay to each key and provide primitives to send a message to a key. Messages are routed to the live node that is currently responsible for the destination key. Keys are chosen from a large space and each node is assigned an identifier (*nodeId*) chosen from the same space. Each node maintains a routing table with nodeIds and IP addresses of other nodes. DHTs use these routing tables to assign keys to live nodes. For instance, in Pastry, a key is assigned to the live node with nodeId numerically closest to the key.

In the simplest case, DHTs can be used to store key-value pairs much like centralized hash tables. Lookup and insert operations can be performed in a small number of routing hops. The overlay network is completely self-organizing, and each node maintains only a small routing table with size constant or logarithmic in the number of participating nodes. DHTs can be used as a platform for a variety of distributed applications, including archival stores [2, 7, 3] and application-level multicast [8, 11].

While there are algorithmic similarities among the proposed DHTs, one important distinction lies in the approach they take to considering and exploiting proximity in the underlying Internet. Considering network proximity is important, because otherwise, a lookup to a key-value pair that is stored on a nearby node may be routed through nodes that are far away in the network (on different continents in the worst case). Three basic approaches have been suggested for exploiting proximity in these DHT protocols [5]:

i) Geographic Layout The nodeIds are assigned in a manner that ensures that nodes that are close in the network topology are close in the nodeId space.

ii) Proximity Routing The routing tables are built without taking network proximity into account but the routing algorithm chooses a nearby node at each hop from among the ones in the routing table. Routing strikes a balance between making progress towards the destination in the nodeId space and chosing the closest routing table entry according to the network proximity.

iii) Proximity Neighbour Selection Routing table construction takes network proximity into account. Routing table entries are chosen to refer to nodes that are nearby in the network topology, among all live nodes with appropriate nodeIds. The distance traveled by messages can be minimized without an increase in the number of routing hops.

Proximity neighbour selection is used in Tapestry and Pastry. The basic Chord and CAN protocols do not consider network proximity at all. However, geographic layout and proximity routing have been considered for CAN [4], geographic layout and proximity neighbor selection are currently being considered for use in Chord [2].

Background

We begin with a brief description of four DHT protocols, Pastry, Tapestry, CAN and Chord.

In Pastry, keys and nodeIds are 128 bits in lengths and can be thought of as a sequence of digits in base 16. A node's routing tables has approximately $log_{16}N$ rows and 16 columns. The 16 entries in row n of the routing table refer to nodes whose nodeIds share the first n digits with the present node's nodeId; the $n+1$th nodeId digit of a node in column m of row n equals m. The column in row n corresponding to the value of the $n+1$'s digits of the local node's nodeId remains empty. Routing in Pastry requires that at each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the present node's id. If no such node is known, 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. The expected number of routing hops is approximately $log_{16}N$.

Tapestry is very similar to Pastry but differs in its approach to locating the numerically closest node in the sparsely populated nodeId space, and in how it manages replication. Pastry uses overlapping sets of neighboring nodes in the nodeId space (leaf sets), both to locate the destination in the final routing hop, and to store replicas of data items for fault tolerance. Tapestry uses a different concept called surrogate routing to locate the destination, and it inserts replicas of data items using different keys. The approach to achieving network locality is very similar in both systems.

The Chord protocol forwards messages in each routing step to a node that is numerically closer to the key. Unlike Pastry and Tapestry, Chord forwards messages only in clockwise direction in the circular id space. To forward messages, each Chord node maintains a finger table, consisting of up to 128 pointers to other live nodes. The ith entry in the finger table of node n refers to a node with the smallest nodeId clockwise from $n + 2^{i-1}$. Note that the first entry points to n's successor, and subsequent entries refer to nodes at repeatedly doubling distances from n. Each node in Chord also maintains a set of clockwise neighbors in the nodeId space (the successor list). The expected number of routing hops in Chord is $\frac{1}{2}log_2N$.

CAN routes messages in a d-dimensional space, where each node maintains a routing table with $O(d)$ entries and any node can be reached in $O(dN^{1/d})$ routing hops. The entries in the routing table are nodes that are neighbours to the current node in the d-dimensional space. Unlike Pastry, Tapestry and Chord, CAN's routing table does not grow with the network size, but the number of routing hops grows faster than $logN$ in this case.

The choice of entries in the routing tables of Chord and CAN is tightly constrained. The CAN routing table entries refer to specific neighboring nodes in each dimension, while the Chord finger table entries refer to specific points in the nodeId space. With Tapestry and Pastry, on the other hand, routing table entries can be chosen arbitrarily from an entire segment of the nodeId space without any impact on the expected number of routing hops. This greatly facilitates proximity based neighbor selection.

Geographic Layout

Geographic layout was explored as one technique to improve routing performance in CAN. The technique attempts to map the d-dimensional space onto the physical network, such that nodes that are neighbours in the d-dimensional space (and therefore in each other's routing tables) are close in the physical network. In one implementation, nodes measure the RTT between themselves and a set of landmark servers to compute the coordinates of the node in the CAN space. This technique can achieve good performance but it has the disadvantage that it is not fully self-organizing; it requires a set of well-known landmark servers. In addition, it may cause significant imbalances in the distribution of nodes in the CAN space that lead to hotspots.

When considering the use of this method in Chord, Tapestry and Pastry, additional problems arise. Whilst geographic layout provides network locality in the routing, it sacrifices the diversity of neighboring nodes in the nodeId space, which has consequences for failure resilience and availability of replicated key/value pairs. Both Chord and Pastry have the property that the integrity of their routing fabric is disrupted when an entire leaf set or successor set fails. Likewise, both protocols replicate key-value pairs on neighboring nodes in the namespace for fault tolerance. With a proximity-based nodeId assignment, neighboring nodes, due to their proximity, are more likely to suffer correlated failures or to conspire.

Proximity Routing

Proximity routing was first proposed in CAN [4]. It involves no changes to routing table construction and maintenance because routing tables are built without taking network proximity into account. But each node measures the RTT to each neighbor (routing table entry) and forwards messages to the neighbor with the maximum ratio of progress in the d-dimensional space to RTT.

Since the number of neighbors is small ($2d$ on average) and neighbors are spread randomly over the network topology, the distance to the nearest neighbor is likely to be significantly larger than the distance to the nearest node in the overlay. Additionally, this approach trades off the number of hops in the path against the network distance traversed at each hop; it may increase the number of hops. Because of these limitations the technique is less effective than geographical layout.

Proximity routing has also been used in a version of Chord [2]. Here, a small number of nodes are maintained in each finger table entry rather than one, and a message is forwarded to the topologically closest node among those entries whose nodeId is closer to but counterclockwise from the message's key. Since all entries are chosen from a specific region of the id space, the expected topological distance to the nearest among the entries is likely to be much larger than the distance of the nearest node in the overlay. Furthermore, it appears that all these entries need to be maintained for this technique to be effective because not all entries can be used for all keys. This increases the overhead of node joins and the size of routing tables.

We conclude that proximity routing affords some improvement in routing performance, but this improvement is limited by the fact that a small number of nodes sampled from specific portions of the nodeId space are not likely to be among the nodes that are closest in the network topology. As we shall see, the structure of the routing tables in Pastry and Tapestry allow a much larger degree of freedom in the selecton of entries, which has a significant impact on the routing performance.

Proximity Neighbour Selection

Tapestry and Pastry's locality properties derive from mechanisms to build routing tables that take network proximity into account. They attempt to minimize the distance, according to the proximity metric, to each of the nodes that appear in a node's routing table, subject to the constraints imposed on nodeId prefixes. Pastry ensures the following invariant for each node's routing table:

Proximity invariant: *Each entry in a node X's routing table refers to a node that is near X, according to the proximity metric, among all live Pastry nodes with the appropriate nodeId prefix.*

As a result of the proximity invariant, a message is normally forwarded in each routing step to a nearby node, according to the proximity metric, among all nodes whose nodeId shares a longer prefix with the key. Moreover, the expected distance traveled in each consecutive routing step increases exponentially, because the density of nodes decreases exponentially with the length of the prefix match. From this property, one can derive two distinct properties of Pastry with respect to network locality:

Total distance traveled The expected distance of the last routing step tends to dominate the total distance traveled by a message. As a result, the average total distance traveled by a message exceeds the distance between source and destination node only by a small constant value.

Local route convergence The paths of two Pastry messages sent from nearby nodes with identical keys tend to converge at a node near the source nodes, in the proximity space. To see this, observe that in each consecutive routing step, the messages travel exponentially larger distances towards an exponentially shrinking set of nodes. Thus, the probability of a route convergence increases in each step, even in the case where earlier (smaller) routing steps have moved the messages farther apart. This result has significance for caching applications layered on Pastry.

The routing algorithms in Pastry and Tapestry allow very effective proximity neigbor selection because there is freedom to choose nearby routing table entries from among a large set of nodes. This leads to very good route locality properties. Moreover, the join protocol allows Pastry to identify appropriate nearby nodes by performaning only a small number of network probes. Analysis and simulations on two Internet topology models presented in [1] confirm this.

CAN also proposed a limited form of proximity neighbor selection in which several nodes are assigned to the same zone in the d-dimensional space. Each node periodically gets a list of the nodes in a neighboring zone and measures the RTT to each of them. The node with the lowest RTT is chosen as the neighbor for that zone. This technique is less effective than those used in Tapestry and Pastry because each routing table entry is chosen from a small set of nodes.

Observations

Geographical layout may be an attractive approach for CAN but it can have a negative impact on the fault-tolerance properties of the other three DHTs. DHTs like Pastry and Chord rely on the diversity of neighboring nodes in the namespace for integrity of the routing fabric and for replication of key-value pairs. Applying geographic layout to these protocols thus raises additional concerns. Even in CAN, constructing the layout in a completely self-organizing fashion may prove difficult. When landmark servers are used, they need to be managed to remain available, and may become bottlenecks under heavy load or denial of service attacks.

Proximity routing requires no changes to routing table construction and maintenance mechanisms whereas proximity neighbor selection requires more expensive mechanisms. In return, proximity neighbor selection is more effective because routing table entries are chosen to be refer to nearby nodes in the first place. However, proximity routing should be sufficient to largely avoid the really bad cases, such as routing to a different continent, then returning.

The effectiveness of proximity neighbor selection depends on the routing algorithm used by the DHT. Algorithms where routing table entries can be selected from among a large set of candidates are able to achieve a lower average hop distance than those that impose more constraints on routing table entries. Therefore, this technique is likely to be more effective in Pastry and Tapestry.

The algorithm described in [1] shows that proximity neighbor selection can be implemented in Pastry with low overhead and that it is very effective at exploiting network proximity; it achieves low delays and rapid route convergence. Experience with applications built on top of Pastry shows that this is important. It is currently an open question whether proximity neighbor selection can be effectively applied to CAN and Chord, or if other, equally effective techniques exist to exploit network proximity in protocols like CAN and Chord.

References

[1] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks, 2002. Submitted for publication.

[2] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[3] J. K. et al. Oceanstore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, November 2000.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, Aug. 2001.

[5] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for dhts: Some open questions. In *Proceedings of IPTPS02*, Cambridge, USA, March 2002. http://www.cs.rice.edu/Conferences/IPTPS02/.

[6] 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)*, Nov. 2001.

[7] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[8] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Third International Workshop on Networked Group Communications*, Nov. 2001.

[9] 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 the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[10] 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.

[11] 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 *Proc. of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001.