PIE: SCALABLE ROUTING IN PEER-TO-PEER NETWORKS

Peter Lichodzijewski
Dalhousie University
piotr@cs.dal.ca

Leigh Wetmore
Dalhousie University
wetmore@cs.dal.ca

A. N. Zincir-Heywood
Dalhousie University
zincir@cs.dal.ca

Abstract

Distributed hash tables (DHTs) are a class of algorithms used to store and retrieve information in peer-to-peer (P2P) systems in a scalable and robust fashion. Pie, a routing protocol for a new DHT framework, is presented. The performance of this algorithm is empirically compared to an existing DHT framework, Chord. Results show that Pie performs better than Chord with respect to the number of logical hops. It also yields a smaller delay and better load balancing. The results are supported by a paired t-test at an alpha level of 0.01.

Keywords: *Peer-to-peer networks; distributed hash tables; distributed protocols; overlay networks.*

1. INTRODUCTION

P2P networks consist of nodes which posses similar capabilities and responsibilities. As a result, there is no centralized control, yielding several advantages such as better scalability and robustness [1]. This however, causes problems in finding information in P2P networks because centralized servers or hierarchies cannot be used.

A proposed approach to retrieving (and storing) information in P2P networks is the distributed hash table (DHT). Each data value is associated with a key, and the set of keys is partitioned among the nodes in the system (i.e., each key is hashed to a node), with a node becoming responsible for the data corresponding to its associated keys. Each node is aware of a handful of other nodes forming an overlay network, and distributed routing algorithms are required for computing nodes responsible for keys using local information only.

Several DHTs have been developed. Pastry [2] and Tapestry [9] use tree-like structures to match progressively larger prefixes of a fixed-length key string [6]. The Content Addressable Network (CAN) represents the key space using a toroidal d-dimensional coordinate space and routes between hyper-rectangles in this space [7]. Finally, Chord uses a circular key space and consistent hashing to provide efficient routing [8].

This work introduces a new DHT routing protocol called Pie. The form of the identifier space and the routing data structures are similar to that of Chord, while the partitioning of the identifier space is similar to that of CAN. The algorithm aims to improve on the routing efficiency of Chord. The actual gap between Chord and Pie is investigated empirically, and a comparative performance between the two algorithms is made. The approach taken is very hands-on, without consideration for dynamic node joins and departures, and its goal is to justify further development of the Pie protocol.

The rest of the paper is organized as follows. Section 2 describes Chord and Pie. Section 3 presents the experiments, results and discussion are presented in section 4, and section 5 concludes the paper

2. THE PROTOCOLS

2.1 Chord

Each key and node in a Chord network is associated with an m-bit identifier. The identifier space is ordered and circular so that 0 follows 2^{m-1}, and all arithmetic is performed modulo 2^m. A key k is assigned to the node whose identifier is equal to or follows k in the identifier space.

For an m-bit identifier space, each node n in a Chord network stores at most m distinct entries in its routing table. For i between 1 and m, the routing table stores the first node that succeeds n by 2^{i-1} or more. The i^{th} entry in the routing tables is referred to as the i^{th} finger of n.

To find a node responsible for a key k, the Chord routing protocol needs to find the successor of k in the identifier circle. It does so iteratively by searching the finger tables for the highest finger that precedes k. The pseudocode for the Chord routing algorithm is shown in Figure 1.

The cost of routing in terms of the number of hops in the overlay network can be derived [8]. It is based on the fact that with every hop, the distance to the ultimate

CCECE 2004- CCGEI 2004, Niagara Falls, May/mai 2004
0-7803-8253-6/04/$17.00 ©2004 IEEE

destination is cut by at least one half, resulting in an O(log(N)) upper bound. The cost of per-node storage is also O(log(N)).

```
// query node n for successor of k
n.find_successor(k)
  n_ = find_predecessor(k);
  return n_.successor;

// query node n for predecessor of k
n.find_predecessor(k)
  n_ = n;
  while(k not in (n_, n_.successor))
    n_ =
n_.closest_preceding_finger(k);
  return n_;

// query node n for closest  finger
// preceeding k
n.closest_preceeding_finger(k)
  for i = m downto 1
    if(n's ith finger in (n, k))
      return ith finger;
  return n;
```

Figure 1. Chord routing protocol.

2.2 Pie

The Pie algorithm is an attempt to improve on the routing performance of Chord. Chord is unidirectional: as one moves in the increasing direction in the identifier space, a Chord node knows less and less about the nodes that fall in that region. In the extreme case, it may take a long time to find an immediately preceding node. Pie attempts to fix this by maintaining information about nodes in both directions and looking both ways during routing.

As in Chord, each key and node in a Pie network is labeled with an identifier from an m-bit space that is ordered and circular. The way in which keys are assigned to nodes, however, is similar to CAN; the identifier space is partitioned into contiguous sectors, and each node is assigned one such sector.

For an m-bit identifier space, a Pie node will store at most 2m distinct entries in its routing table. For i between 1 and m, the routing table of a node n stores the node responsible for keys $n + 2^{i-1}$ and $n - 2^{i-1}$. The node responsible for key $n + 2^{i-1}$ is n's i^{th} finger going clockwise (CW), and the node responsible for $n - 2^{i-1}$ is n's i^{th} finger going counter-clockwise (CCW). In addition, the bounds of each key's sector are also stored.

If a Pie node does not know the identity of a node responsible for a key, it checks all entries in its routing table and forwards the request to the node that is closest to the requested key. The pseudo code for the Pie protocol is shown in Figure 2.

```
// find the node responsible for k
n.find_owner(k)
  n_ = n;
  while(k not in n_.get_sector())
    n_ = find_closest_finger(k);
  return n_;

// find the node closest to k
n.find_closest_finger(k)
  d_cw = finger_cw(1);
  d_ccw = finger_ccw(1);
  for i = 2 upto m
    if(d_cw > finger_cw(i).distance(k))
      d_cw = finger_cw(i);
    if(d_ccw > finger_ccw(i).distance(k))
      d_ccw = finger_cw(i);
  if(d_cw < d_ccw)
    return d_cw;
  return d_ccw;
```

Figure 2. Pie routing protocol.

The find_owner() procedure iteratively queries nodes closer and closer to the requested key k until the node whose sector contains k is found. The second procedure, when called at a node n, returns the node in n's finger table that is closest to k.

In Pie, the distance from a node n to a key k is the minimum distance from k to any identifier in n's sector. If n's sector is the range [l, u], the distance is defined as the distance from l to k or u to k, whichever is smaller. If k falls in [l, u], the distance is zero. Since the identifier space is circular, the shorter distance between two points is always taken.

Partitioning of the identifier space in Pie is adapted from CAN. A joining node with identifier n first finds the node p that is already part of the network and is responsible for the identifier n. Node p's sector is then split into two, one for each of the two nodes. Splitting is done so as to make the two new sectors as similar in size as possible, making sure that both n and p fall in their own respective sectors.

3. EXPERIMENTS

In order to compare Chord and Pie empirically, both algorithms were implemented using a discrete event simulator [3] and tested on a series of lookup requests. A recursive approach, whereby each node along the path forwards the request to the next, was taken.

The protocols were tested on eight different cases each with a different number of physical nodes (N),

physical edges (E), and different connectivity (maximum number of neighbours per node, NBR). In each case, eighty percent of the physical nodes were implemented at the logical layer, and twenty runs were made to allow for proper statistical testing.

Lookup requests were made using an exponential distribution with a mean of 100 milliseconds. Throughput at the physical layer was set at 10 Mbps, and propagation delay was set at 0.03 seconds. Both protocols used a 10-bit identifier space.

Note that in each case all parameters, including the set of node identifiers and key requests, were kept constant. The only variable was the routing protocol.

4. RESULTS AND DISCUSSION

Figures 3 through 6 show the results with respect to the average number of physical hops and logical hops, the delay per request, and the standard deviation of the number of keys for which each node is responsible. The numbers are averaged over twenty runs in each case.

For each metric and test case, a paired t-test over the twenty runs was used to determine whether there was a significant difference between Pie and Chord. Using an alpha level of 0.01, it was determined that Pie performs significantly better than Chord with respect to the number of logical hops, the delay, and the variance in the number of keys stored for *all* test cases. For the number of physical hops the results are inconclusive for all cases.

From the results it is clear that, requiring half as many, Pie performs better than Chord with respect to the number of logical hops. This is intuitive given Pie's bidirectionality. It may also account for longer delays in Chord where more processing has to be done. Similar results for the number of physical hops suggest that similar paths are taken by both protocols at the physical network layer.

Chord is a form of consistent hashing and so guarantees a certain level of load balancing [4]. It is encouraging that Pie appears to do even better as suggested by the keys per node metric.

Note that even though join operations were not implemented explicitly, the state of the network (i.e., routing tables) on which tests were performed would have been the same had they been.

In order for Pie to be useful, it must be able to deal with concurrent node joins and departures. This type of functionality has already been designed for Chord, where performance is sacrificed for correctness, and resolution of requests is guaranteed by maintaining correct successor pointers [8]. One mechanism for Pie to be able to deal with concurrent node joins and departures is similar to that of Chord.

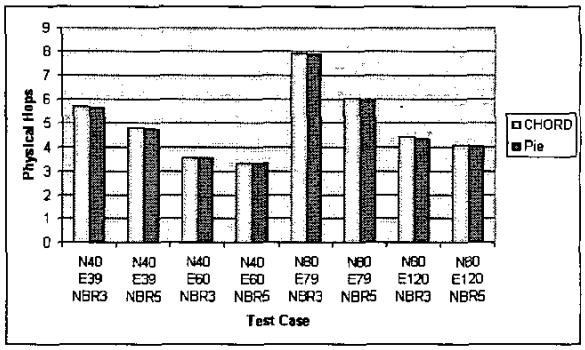

Figure 3. Physical hops.

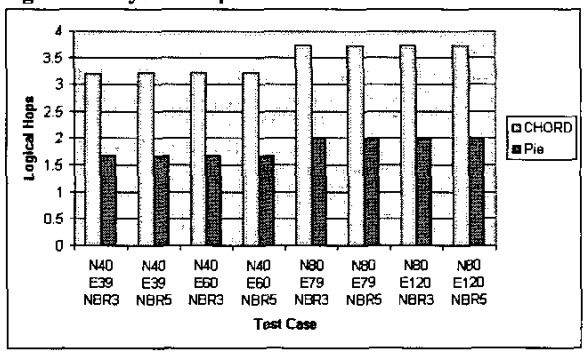

Figure 4. Logical hops.

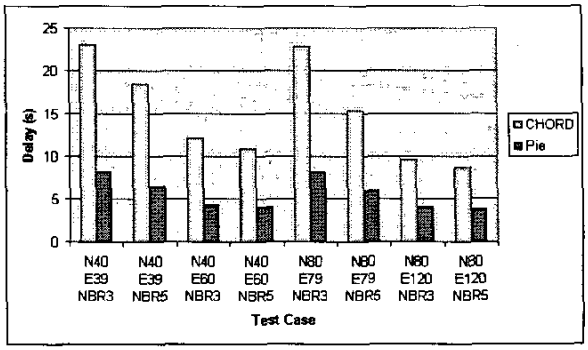

Figure 5. Delay in seconds.

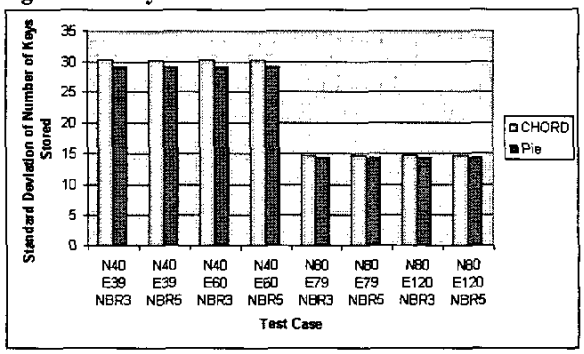

Figure 6. Variation in number of keys per node.

In Pie this functionality may possibly be implemented as follows. A Pie node would keep track of the nodes that are responsible for the sectors adjacent to its own, i.e., its neighbours. A Pie node n would join the network by querying an existing node for n's neighbours. Other nodes would find out about n through a periodic stabilization process. During this process, each node would verify that its neighbours are correct by making sure that it is its neighbours' neighbour. The network would be unaware of a node n until n performs this stabilization process, notifying its neighbours of its existence.

This mechanism relies on the correct identification of neighbouring nodes. Thus, even if the rest of the finger tables are wrong, lookup requests can still be resolved by way of a linear scan of neighbours. The finger tables themselves would periodically be refreshed to keep Pie's performance at a high level.

To protect the system against node failures, several candidate neighbour nodes could be maintained, or several nodes could be assigned to each sector [7]. Requests to failed nodes would time out and be redirected through an alternate route.

Pie performs better than Chord with respect to routing, but this comes at a cost of having to store more information at each node. In addition, to make a more solid comparison of the two algorithms, dynamic node joins and departures need to be included in the simulation. Future work on Pie should focus on extending its functionality to include concurrent node joins and departures as well as developing theoretical results in support of the empirical results.

5. CONCLUSIONS

This work presented a new DHT algorithm called Pie. Based on CAN and Chord, the main purpose of Pie is to improve on the routing efficiency of Chord. To this end, several simulations were conducted using a discrete event simulator. The results of these simulations showed that Pie is significantly better than Chord with respect to the number of logical hops and the delay required to resolve a lookup request, and in balancing the number of keys for which each node in the P2P network is responsible. These results warrant further development of Pie, and in particular, functionality that would enable it to deal with concurrent node joins and departures.

Acknowledgements

This work was conducted while Leigh Wetmore was funded by an NSERC graduate award and while Peter Lichodzijewski was funded by an NSERC graduate award and the Izaak Walton Killam Memorial Scholarship.

References

[1] Balakrishnan, H., Kaashoek, M., Karger, D., Morris, R., Stoica, I., "Looking up data in P2P systems," Communications of the ACM, 46(2), pp 43-48, 2003.

[2] Druschel, P., Rowstron, A., "Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems," Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329-350, 2001.

[3] JavaSIM, http://javasim.ncl.ac.uk, Accessed February 2, 2003.

[4] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., Panigrahy, R., "Consistent hashing and random trees: Distributed caching protocols and relieving hot spots on the world wide web," Proceedings of the ACM Symposium on Theory of Computing, pp. 654-663, 1997.

[5] Liben-Nowell, D., Balakrishnan, H., Karger, D., "Analysis of the evolution of peer-to-peer systems," Proceedings of the ACM Conference on Principles of Distributed Computing, pp. 233-243, 2002.

[6] Plaxton, C., Rajaraman, R., Richa, A., "Accessing nearby copies of replicated objects in a distributed environment," Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pp. 311-320, 1997.

[7] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., "A scalable content addressable network," Proceedings of ACM SIGCOMM 2001, pp. 161-172, 2001.

[8] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H., "Chord: a scalable peer-to-peer lookup protocol for internet applications," Proceedings of the ACM SIGCOMM 2001, pp. 149-160, 2001.

[9] Zhao, B., Kubiatowicz, J., Joseph, A., "Tapestry: an infrastructure for fault-tolerant wide-area location and routing," Technical Report UCB/CSD-01-1141, U.C. Berkeley, 2001.