PathFinder: Efficient Lookups and Efficient Search in Peer-to-Peer Networks

Dirk Bradler[1], Lachezar Krumov[1], Max Mühlhäuser[1], and Jussi Kangasharju[2]

[1] TU Darmstadt, Germany
{bradler,krumov,max}@cs.tu-darmstadt.de
[2] University of Helsinki, Finland
jussi.kangasharju@cs.helsinki.fi

Abstract. Peer-to-Peer networks are divided into two main classes: unstructured and structured. Overlays from the first class are better suited for exhaustive search, whereas those from the second class offer very efficient key-value lookups. In this paper we present a novel overlay, PathFinder, which combines the advantages of both classes within one single overlay for the first time. Our evaluation shows that PathFinder is comparable or even better in terms of lookup and complex query performance than existing peer-to-peer overlays and scales to millions of nodes.

1 Introduction

Peer-to-peer overlay networks can be classified into *unstructured* and *structured* networks, depending on how they construct the overlay.

In an unstructured network the peers are free to choose their overlay neighbors and what they offer to the network.[1] In order to discover if a certain piece of information is available a peer must somehow search through the overlay. There are several implementations of such search algorithms. The original Napster used a central index server, Kazaa relied on a hybrid network with supernodes and the original Gnutella used a decentralized flooding of queries [4]. The BubbleStorm network [5] is a fully decentralized network based on random graphs and is able to provide efficient exhaustive search.

Structured networks, on the other hand, have strict rules about how the overlay is formed and where content should be placed within the network. Structured networks are also often called distributed hash tables (DHT) and the research world has seen several examples of DHTs [3,7]. DHTs are very efficient for simple key-value lookups. Because objects are addressed with their unique names, searching in a DHT is hard to be made more efficient [6]. However, wildcard searching and complex queries either impose extensive complexity and costs in terms of additional messages or are not supported at all.

Given the attractive properties of both these different network structures: it is natural to ask the question: *Is it possible to combine these two properties in*

[1] In this paper we focus on networks where peers store and share content, e.g., files, database items, etc.

M.K. Aguilera et al. (Eds.): ICDCN 2011, LNCS 6522, pp. 77–82, 2011.

one single network? Our answer to this question is PathFinder, a peer-to-peer overlay which combines an unstructured and a structured network in a single overlay. PathFinder is based on a random graph which gives it short average path length, large number of alternative paths for fault tolerable, highly robust and reliable overlay topology. Our main contribution is the efficient combination of exhaustive searching and key-value lookups in a single overlay.

The rest of this paper is organized as follows. In Section 2 we present an overview of PathFinder. Section 3 compares it to existing P2P overlays and we conclude in Section 4. Due to space limitations, the reader is referred to [1] for technical aspects such as node join and leave, handling crashed nodes, and network size adaptation. In [1] an extensive evaluation of PathFinder under churn and attacks is also presented.

2 PathFinder Design

In this section we present the system model and preliminaries of PathFinder. We also describe how the basic key-value lookup and exhaustive search work. For further basic operations like node join/leave, handling crashed nodes, see [1].

2.1 Challenges

We designed PathFinder to be fully compliant with the concept of BubbleStorm [5], namely an overlay structure based on random graphs. We augment the basic random graph with a deterministic lookup mechanism (see Section 2.4) to add efficient lookups into the exhaustive search provided by BubbleStorm. The challenge and one of the key contributions of this paper is developing a deterministic mechanism for exploiting these short paths in implement DHT-like lookups.

2.2 System Model and Preliminaries

All processes in PathFinder benefit from the properties of its underlying random graph and the routing scheme built on top of it.

PathFinder construction principle. The basic idea of PathFinder is to build a robust network of virtual nodes on top of the physical peers (i.e. actual physical nodes). Routing among peers is carried out in the virtual network. The actual data transfer still takes place directly among the physical peers. PathFinder builds a random graph of virtual nodes and then distributes them among the actual peers. At least one virtual node is assigned to each peer. From the routing point of view, the data in the network is stored on the virtual nodes.

When a peer B is looking for a particular piece of information it has to find a path from one of its virtual nodes to the virtual node containing the requested data. Then B directly contacts the underlying peer A which is responsible for the targeted virtual node. B retrieves the requested data directly from A. This process is described in detail in Section 2.4.

It is known that the degree sequence in a random graph is Poisson distributed. We need two pseudorandom number generators (PRNG) which initialized with the same ID always produce a deterministic sequence of numbers. Given a number c, the first generator returns Poisson distributed numbers with mean value c. The second PRNG given a node ID produces a deterministic sequence of numbers which we use as IDs for the neighbors of the given node.

The construction principle of PathFinder is as follows. First we fix a number c (see [1] on how to chose c according to the number of peers and how to adapt it once the network becomes too small/large). Then, for each virtual node we determine the number of neighbors with the first number generator. The actual nodes IDs to which the current virtual node should be connected are chosen with the second number generator. The number generator is started with the ID of the virtual node. The process can be summarized in the following steps:

1. The underlying peer determines how many virtual nodes it should handle. See [1] for details.
2. For every virtual node handled by the peer:
   (a) The peer uses the poisson number generator to determine the number of neighbors of the current virtual node.
   (b) The peer then draws as many pseudo random numbers according to the number drawn in the previous step.
   (c) The peer selects the virtual nodes with IDs matching to those numbers as neighbors for its current virtual node.

The construction mechanism of PathFinder allows the peers to build a random graph out of their virtual nodes. It is of crucial importance that a peer only needs a PRNG to perform that operation. There is no need for network communication. Similarly, *any peer* can determine the neighbors of *any virtual node*, simply seeding the pseudo random generator with the corresponding ID.

Now we have both, a random graph topology suited for exhaustive search and a mechanism for each node to compute the neighbor list of any other node. i.e. DHT-like behavior within PathFinder.

Routing table example of PathFinder. Figure 1 shows a small sample of PathFinder with a routing table for the peer with ID 11. The random graph has 5 virtual nodes (1 through 5) and there are 4 peers (with IDs from 11 through 14). Peer 11 handles two virtual nodes (4 and 5) and all the rest of the peers have 1 virtual node each. The arrows between the virtual nodes show the directed neighbor links.

Each peer keeps track of its own outgoing links as well as incoming links from other virtual nodes. A peer learns the incoming links when the other peers attempt to connect to it. Keeping track of the incoming links is strictly speaking not necessary, but makes key lookups much more efficient (see Section 2.4). The routing table of peer marked as 11 therefore consists of all outgoing links from its virtual nodes 4 and 5 and the incoming link from virtual node 3.

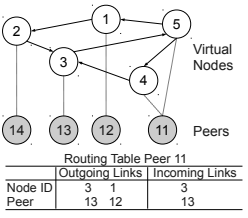

Fig. 1. A small example of PathFinder

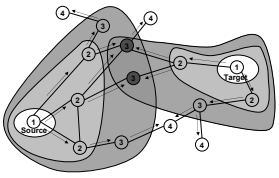

Fig. 2. Key lookup with local expanding ring search from source and target

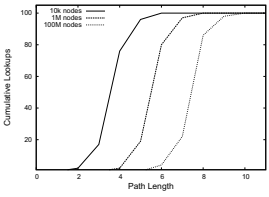

Fig. 3. Distribution of complete path length, 5000 key lookups with $c = 20$

2.3 Storing Objects

An object is stored on the virtual node (i.e. on the peer responsible for the virtual node) which matches the object's identifier. If the hash space is larger than the number of virtual nodes, then we map the object to the virtual node whose identifier matches the prefix of the object hash.

2.4 Key Lookup

Key lookup is the process when a peer contacts another peer possessing a given data of interest. Using the structure of the network, the requesting peer traverses only one single and usually short path from itself to the target peer.

Key lookup is the main function of a DHT. In order to perform quick lookups, the average number of hops between peers as well as the variance needs to be kept small. We now show how PathFinder achieves efficient lookups and thus behaves as any other DHT. Suppose that peer A wants to retrieve an object O. Peer A determines that the virtual node w is responsible for object O by using the hash function described above. Now A has to route in the virtual network from one of its virtual nodes to w and directly retrieve O from the peer responsible for w.

Denote with V the set of virtual nodes managed by the peer A. For each virtual node in V, A calculates the neighbors of those nodes. (Note that this calculation is already done, since these neighbors are the entries in peer A's routing table.) A checks if any of those neighbors is the virtual node w. If yes, A contacts the underlying peer to retrieve O. If none of peer A's virtual node neighbors is responsible for O, A calculates the neighbors of all of its neighbors, i.e. its second neighbors. Because the neighbors of each virtual node are pre-known (see Section 2.2), this is a simple local computation. Again, peer A checks if any of the new calculated neighbors is responsible for O. If yes, peer A sends its request to the virtual node whose neighbor is responsible for O. If still no match is found, peer A expands its search by calculating the neighbors of the nodes from the previous step and checks again. The process continues until a match is found. A may have to calculate several neighbors, but a match is guaranteed.

Because peer A is able to compute w's neighboring virtual nodes, A can expand the search rings *locally* from both the source and target sides, which is called forward and backward chaining. In every step the search depth of the source and target search ring is increased by one. In that way the number of rings around the source are divided between the source itself and the target. This leads to exponential decrease in the number of IDs that have to be computed.

We generated various PathFinder networks from 10^3 up to 10^8 nodes with average degree 20. In all of them we performed 5000 arbitrary key lookups. It turned out that, expanding rings of depth 3 or 4 (i.e., path length between 6 and 8) is sufficient for a successful key lookup, as shown in Figure 3.

2.5 Searching with Complex Queries

PathFinder supports searching with complex queries with tunable success rate almost identical to BubbleStorm [5]. In fact, since both PathFinder and BubbleStorm are based on random graphs, we implemented the search mechanism of BubbleStorm directly into PathFinder. In BubbleStorm both data and queries are sent to some number of nodes, where the exact number of messages depends on how we set the probability of finding the match. We use exactly the same algorithm in PathFinder for searching and the reader is referred to [5] for details.

3 Comparison and Analysis

Most DHT overlays provide the same functionality, since they all support the common interface for key based routing. The main differences between various DHT implementations are average lookup path length, resilience to failures, and load balancing. In this Section we compare PathFinder to established DHTs.

The lookup path length of Chord is well studied: $L_{avg} = \frac{\log(N)}{2}$. The maximum path length of Chord is $\frac{\log(N)}{\log(1+d)}$. The average path length of PathFinder is $\frac{\log(N)}{\log(c)}$, where c is the average number of neighbors. The path length of the Pastry model can be estimated by $\lceil \log_{2^b}(N) \rceil$ [3], where b is a tunable parameter. The Symphony overlay is based on a small world graph. This leads to key lookups in $O(\frac{\log^2(N)}{k})$ hops [2]. The variable k refers only to long distance links. The actual amount of neighbors is indeed much higher [2]. The diameter of CAN is $\frac{1}{2}dN^{\frac{1}{d}}$ with a degree for each node $2d$, with a fixed d. With large d the distribution of path length becomes gaussian, like Chord.

We use simulations to evaluate the practical effects of the individual factors. Figure 4 shows the results for a 20,000 nodes network. We perform 5,000 lookups among random pairs of nodes and measure the number of hops each DHT takes to find the object. Figure 5 displays the results. Note that PathFinder results come from *actual simulation*, not analytical calculations.

PathFinder also inherits the exhaustive search mechanism of BubleStorm. Hence, as an unstructured overlay it performs identical to BubleStorm and the reader is referred to [5] for thorough comparison to other unstructured systems.

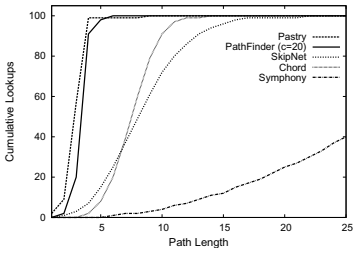

Fig. 4. Average number of hops for 5,000 key lookups in different DHTs

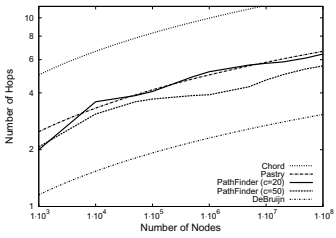

Fig. 5. Average number of hops for different DHTs measured analytically. Numbers for PathFinder are simulated.

4 Conclusions

In this paper we have presented PathFinder, an overlay which combines efficient exhaustive search and efficient key-value lookups in the same overlay. Combining these two mechanisms in the same overlay is very desirable, since it allows efficient and overhead-free implementation of natural usage patterns. PathFinder is the first overlay to combine exhaustive search and key-value lookups in an efficient manner.

Our results show that PathFinder has performance comparable or better to existing overlays. It scales easily to millions of nodes and its key lookup performance is in large networks better than in existing DHTs. Because PathFinder is based on a random graph, we are able to directly benefit from existing search mechanisms (BubbleStorm) for enabling efficient exhaustive search.

References

1. Bradler, D., Krumov, L., Kangasharju, J., Weihe, K., Mühlhäuser, M.: Pathfinder: Efficient lookups and efficient search in peer-to-peer networks. Tech. Rep. TUD-CS-2010872, TU Darmstadt (October 2010)
2. Manku, G., Bawa, M., Raghavan, P.: Symphony: Distributed Hashin In A Small World. In: Proc. 4th USENIX Symposium on Internet Techn. and Systems (2003)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Liu, H. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
4. Steinmetz, R., Wehrle, K. (eds.): Peer-to-Peer Systems and Applications. LNCS, vol. 3485. Springer, Heidelberg (2005)
5. Terpstra, W., Kangasharju, J., Leng, C., Buchmann, A.: Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In: Proc. SIGCOMM, pp. 49–60 (2007)
6. Yang, Y., Dunlap, R., Rexroad, M., Cooper, B.: Performance of full text search in structured and unstructured peer-to-peer systems. In: Proc. IEEE INFOCOM (2006)
7. Zaho, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Comp. 74 (2001)