Exploiting Local Popularity to Prune Routing Indices in Peer-to-Peer Systems

Stephane Bressan
Department of Computer Science
National University of Singapore
steph@comp.nus.edu.sg

Achmad Nizar Hidayanto Zainal A. Hasibuan
Faculty of Computer Science
University of Indonesia
{nizar, zhasibua}@cs.ui.ac.id

Abstract

Routing in unstructured peer-to-peer systems relies either on broadcasting (also called flooding) or on routing indices. An approach using routing indices is only scalable if the routing indices are of manageable size. In this paper, we present a strategy to prune routing indices based on popularity of resources. Routing indices maintain routing information for queries to the most popular resources, leaving queries to other resources to be routed randomly. The popularity of resources at each node of a network is learnt by each routing index, by means of a replacement strategy. We compare the performance of the local popularity method against that of the global popularity method in pruning routing indices in both static and dynamic environments. We compare the effectiveness and the efficiency of two standard replacement strategies: Least Frequently Used (LFU) and Least Recently Used (LRU). Our results confirm the efficiency and effectiveness of pruning routing indices based on the local popularity of resources in unstructured peer-to-peer networks.

1 Introduction

Peer-to-peer technology can be divided into two main classes according to how nodes are organized and how they acquire resources, namely structured and unstructured peer-to-peer architectures. In structured architectures such as [11, 10, 3], nodes are connected to each other, and resources or records are moved according to some mapping techniques (usually hashing).

In unstructured architectures such as [13, 2], nodes are connected to each other as they join the network, and resources are not moved but hosted on site. Hence, searching for a resource requires either a broadcast request or use routing information. The design of routing indices is therefore critical to achieve an efficient, effective and scalable system.

Our past work in the area includes [7], where we presented a method of pruning routing indices based on global popularity. In this paper, we widen our scope and consider a promising alternative pruning method. The contribution of this paper is two-fold:

1. We propose a general approach to prune routing indices by learning the popularity of resources in the local context.

2. We provide a comparative evaluation of performance between the local popularity method and the global popularity method in pruning routing indices in both static and dynamic environments.

In the next section, we present a brief overview of the main related work on search and routing strategies in unstructured peer-to-peer systems. As with our previous work in the area (see [7] and [1]), we adopt a reinforcement learning approach called Q-routing to learn routing information. We briefly recall the principles underlying reinforcement learning in Section 3. In Section 4, we describe routing indices and routing strategies, and propose our new strategy to prune routing indices. In Section 5, we evaluate and analyze the performance of our proposed approach in both static and dynamic environments. We present our conclusions in the last section.

2 Related Work

We outline here the respective architectures of the two main operational unstructured peer-to-peer systems: Gnutella [13] and Freenet [2]. Gnutella is a decentralized file-sharing system whose participants form a virtual network communicating from peer to peer. They use the Gnutella protocol, which consists of five basic message types: ping, pong, query, queryHit and push. These messages are routed to their destinations by constrained broadcast, which works by decrementing a message's time-to-live field.

Freenet, an anonymous peer-to-peer storage system, is decentralized and symmetric, and it automatically adapts

when nodes leave or join. Freenet does not assign responsibility for documents to specific servers; instead, it performs lookups in the form of searches for cached copies. This allows Freenet to provide a degree of anonymity, but prevents it from guaranteeing retrieval of existing documents or providing low bounds on retrieval costs.

The authors of [4] introduced an interesting alternative to constrained broadcasting for routing requests in unstructured peer-to-peer systems, namely routing indices. Routing indices are tables located and used at individual nodes of a network to forward queries to neighbors that are likely to have the answers needed. When a query arrives at a node, the node uses its routing index to select neighbor nodes and forwards the query to them. The authors of [4] proposed three different types of routing indices: compound, hop-count and exponential. Compound routing indices provide the number of resources that can be obtained when a query is forwarded to a selected neighbor. They provide only information on the number of resources, and no information on distance. Hop-count routing indices provide distance information in addition to resource number. Likewise, exponential routing indices provide both kinds of information, but they present the information in a different way.

Routing indices are constructed when local information is propagated to neighboring nodes at a fixed interval or when changes in local indices have reached a certain threshold. In [4], resources in a network are classified into different classes, and indexing is done with these classes. While this reduces the size of the index, it prevents finer granularity. We have shown in [7] and [1] that routing indices can be made adaptive by means of a reinforcement learning approach to their construction and maintenance.

3 Background

3.1 Peer-to-Peer Search and Routing Indices

The challenge in the routing process is to efficiently route a query to the next node in minimal time. As we have mentioned, routing indices are often used, and they are maintained at each node, storing the costs of routing requests for resources via each of its neighbors. In our early work [7], we developed an adaptive routing mechanism that relies on the Q-Routing algorithm for learning routing information. Each node in the network maintains a routing index called the Q-table; for each neighbor node, the table stores the estimated time that a query may take to reach a node that hosts the requested resource. We have shown that the algorithm is effective and leads to converging and efficient routing time.

In [7], we also experimented with various techniques for pruning routing indices. We showed that the best criteria is popularity of resources (as opposed, for instance, to distance of resource location to the local node). Yet, as we noted in that work, we assumed that the distribution of popular resources was known and global. This is the hypothesis that we challenge in this paper.

In [1], we extended the adaptive routing approach to keyword search. In the approach, reinforcement learning estimates both routing time and relevance. Routing becomes a compromise between efficiency and effectiveness. We showed how users could tune this compromise, and we proposed a device that could automatically tune the compromise as a function of network resources.

In this paper, we focus on pruning strategies for routing indices. These pruning strategies are very important to decreasing the size of routing indices at each node. We define the routing and search problem in peer-to-peer networks as follows: We consider a set of nodes and their routing indices, $P = (p_1, ri_1), ..., (p_n, ri_n)$, and a set of resources, $R = r_1, ..., r_m$, located on the nodes and identified by the set of keys $K = k_1, ..., k_m$. Given a request of the form $locate(k_i)$, i.e., a request to locate resource r_i, a node receiving the request must make a local decision as to which of its neighbors it should forward the request to, based on information from its routing index.

3.2 Reinforcement Learning

A reinforcement learning agent[12] learns from its interactions with its environment. In a given state, the reinforcement learning agent selects the next action according to a model which constantly constructs and updates based on the rewards that it receives when trying different actions. A routing agent could, for instance, favor an action that yields minimum estimated time to destination. Notice that in order to continuously learn about its environment, the agent must occasionally depart from its base strategy and try random actions.

Reinforcement learning has been successfully applied to many domains, including network routing. Numerous forms and variants of reinforcement learning have been proposed for various network routing problems: Q-Routing [8], Dual Q-Routing [6], Confidence-based Q-Routing and Dual Confidence-based Q-Routing [5], etc. The experiments in [5] show that the confidence-based variant yields the best performance among all of them, but it also requires large storage. For simplicity and scalability, we adopt Q-Routing in this study.

In Q-Routing, if a node receives a request for a resource that it is hosting, it provides the resource to the original requester. If it does not have the resource, it forwards the request according to the reinforcement learning strategy: It adopts the forward propagation strategy to estimate the status of the network by maintaining a table called the Q-table, which contains Q-value $Q(n, r)$ which quantifies the

expected reinforcement when a request for a resource r is sent to n.

Such algorithms can outperform shortest-path algorithms such as Bellman-Ford for they are able to estimate routing time with traffic taken into account.

4 Design

At each node of a peer-to-peer network, we maintain a routing index. The values it contains are estimates of the routing time of a request to the corresponding resource. As we have mentioned, we use reinforcement learning, specifically Q-routing, to maintain values stored in the routing indices and to route requests.

As storage capacity at each node is limited, a routing index cannot store information about all possible requests. This does not matter since some requests are so infrequent (i.e., the resources are not popular) that a longer service time would not impact the overall performance of the system. Thus, we can view routing indices as caches. We may want to store only important information that will be accessed frequently due to the limited size of the cache relative to the amount of information available for storage. We can use a number of page replacement algorithms to control information that should be moved in and out of the cache. For that purpose, we adopt a mechanism to learn the popularity of resources in the local context.

In the next subsections, we will briefly describe our routing strategy design and propose our pruning strategy that is to be incorporated into the routing strategy.

4.1 Routing Strategy

Q-Routing [8] is an adaptive routing strategy that uses Q-learning, a reinforcement learning algorithm. Q-Routing has been examined in a number of network routing studies and they show that it is able to adapt to network traffic.

In Q-learning [12], states and possible actions in a given state are discrete and finite in number. The method uses a set of values termed Q-values for learning. Each Q-value is denoted by *Q(s, a)* where s is the given state and *a* is the action taken. In general, Q-value represents the expected reinforcement when an action a is being taken in a state s.

Given the current state of a network, the Q-Routing algorithm learns and finds an optimal routing policy from the Q-values distributed over all nodes in the network. Each node p in the network has its own view of the state of the network through a table that stores all the Q-values, Q_p*(p', r)*, where $r \in R$ is a set of resource objects in the peer-to-peer network and $p' \in N(p)$ is a set of all neighbors of node p. Q_p*(p', r)* is defined as the best estimated time that a query takes to reach a node that hosts a resource r from a node p through its neighbor p'.

The Q-value Q_p*(p', r)* can be represented by the following mathematical equation:

$$Q_p(p', r) = q_{p'} + \delta + Q_{p'}(p'', r) \quad (1)$$

From the equation, we can derive that the minimum time needed to locate a resource *r* in the peer-to-peer network from a node *p* through its neighbor *p'* is affected by three components:

1. The time the query spends at node p', $q_{p'}$.
2. The transmission delay between p and p'.
3. The best estimation time for $Q_{p'}(p'', r)$.

Once we are clear what the Q-values stored at each node stand for, it is easy to see how we can use them to make locally optimal routing decisions. When a node p receives a query $q(s, r)$ looking for a resource r, node p will look up its Q-value table $Q_p(*, r)$ to select the neighboring node p' with the minimum $Q_p(p', r)$ value. With this mechanism being used at every node in the network, the query will be answered in minimal time.

4.2 Updates of Q-value

The performance of the routing strategy depends on all Q-values in the path taken by the query. The closer these Q-values are to the actual time needed, the closer the routing strategy will reflect actual optimal time. Therefore, the process of updating these Q-values is very important.

Recall that when a node p receives a query $q(s, r)$, it finds node p' in its neighbor list such that Q_p fulfills the following condition:

$$Q_p(p', r) = \min_{p' \in N(p)} Q_p(p', r) \quad (2)$$

Once p' is chosen, p forwards the query $q(s, r)$ to p'. Upon receiving the query, p' first searches if it has the requested resource r. If r is not in the shared list of p', it forwards the query to its neighbor. For both cases, p' has to send the Q-value back to p. In the first case, since the query can be answered by p', no further routing of the query $q(s, r)$ is needed, and the Q-value returned to p is 0. In the second case, the Q-value, $Q_{p'}(p'', r)$, will be returned according to equation 2.

The Q-value returned by p' is essentially its estimation of how long it takes a query to be answered after p' has processed the query. The estimation returned excludes the queuing time at p' and the transmission delay, δ, for the query to travel from p to p'. Based on the estimated value, node p will calculate the new Q-value for $Q_p(p', r)$ with the following equation:

$$Q_p(p', r)^{new} = Q_p(p', r)^{old} + l(Q_{p'}(p'', r)^{new} + q_{p'} + \delta - Q_{p'}(p'', r)^{old}) \quad (3)$$

where l is the learning rate with $0 \leq l \leq 1$. When l is set to 1, the equation becomes:

$$Q_p(p', r) = Q_{p'}(p'', r) + q_{p'} + \delta \quad (4)$$

However, since the new Q-value sent back by p' is the propagation of estimated values, the Q-value is not necessarily accurate. Therefore, in order to reflect the possible error caused by estimation in the learning process, the learning rate should always be set to a value less than 1 (eg., 0.7 has been a suggested learning rate in previous network routing studies).

4.3 Pruning of Routing Indices

Figure 1 depicts the structure of a routing index that stores Q-values. Each resource r is a record row in the

Resources	Neighbour Q-value			
	p1	p2	...	pn
r1	12	6	...	8
r2	10	12	...	7
r3	4	10	...	9
rn	3	4		9

Figure 1. Q-value table for node p

routing index. For each resource, there is a corresponding Q-value for each neighbor. Since the number of resources in the network determines the number of rows, the size of the routing index is proportional to the resources shared in the network and it is potentially big. In this work, we study mechanisms to maintain the size of the Q-value table; we propose the following rules for the table:

- Stores only n entries that are most recently requested
- Stores only n entries that are most frequently requested

The proposed rules are an adaptation of the mechanism in page replacement algorithms which we will explain in the following subsections.

4.3.1 Global Popularity Method

This pruning method is based on resource popularity. It indexes resources that are most frequently requested in the network, and it assumes that we always know the popularity of all resources. We can operationalize this assumption by generating a popularity list using Zipfian distribution. Those resources not listed in the popularity list would be ignored.

4.3.2 Most Recently Used Method

This method is based on the Least Recently Used (LRU) algorithm. Each node learns the local popularity of resources by examining queries that it receives. Each routing index only stores n resources which are most recently requested, and their corresponding Q-values. When a node receives a request on resource r, its routing index is updated by the replacement of a resource that has been least recently requested with r and its corresponding Q-values.

4.3.3 Most Frequently Used Method

This method is based on the Least Frequently Used (LFU) algorithm. Each node learns the local popularity of resources by examining queries that it receives. Each routing index only stores n resources which are most frequently requested, and their corresponding Q-values. When a node receives a request for resource r, its routing index is updated with the replacement of a resource that has been least frequently requested with r and its corresponding Q-values. We add a column to the routing index to store frequency information. The value in the frequency column is set to 1 when a resource is put into the routing index for the first time or when it replaces the least frequently used resource. The value is incremented by 1 if the resource has already been residing in the routing index.

5 Performance Analysis

We simulated a peer-to-peer network to study the performance of our pruning strategies. We repeated the experiments five times to compute the average routing time.

The network was a randomly re-wired ring lattice. A ring lattice is a graph with n vertices, each of which is connected to its nearest k neighbors. In the experiments, we used n and k values of 36 and 4 respectively. The value of n could be scaled as shown in [7]. The rewiring was done by randomly choosing m vertices to connect with each other. We used an m value of 10% of n. This relatively small number compared to n was meant to reflect the small world effect [9].

We randomly assigned d resources to n nodes according to uniform distribution. We used a d value of 100. The workload was determined by a Zipfian distribution corresponding to popularity of resources. Requests for resources were generated according to this distribution. The originator node of the requests were chosen randomly.

Figures 2[1] to 5 show the experiment results from a static peer-to-peer environment where no nodes were joining or leaving. We first compare the performances of the global popularity method, the local popularity method with LRU,

[1] The graphs should be viewed in color.

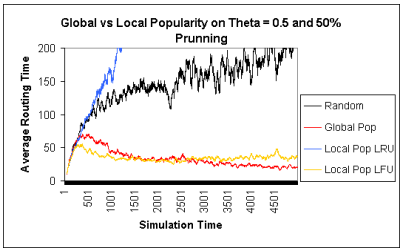

Figure 2. Result on 50% pruning with θ = 0.5

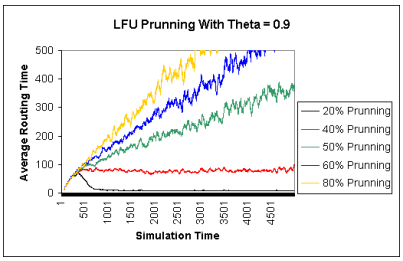

Figure 3. Result using LFU with θ = 0.9

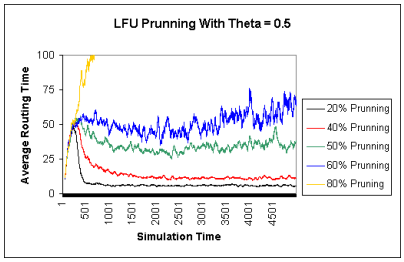

Figure 4. Result using LFU with θ = 0.5

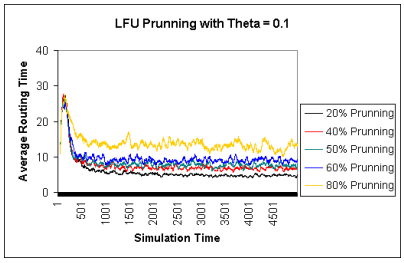

Figure 5. Result using LFU with Zipf θ 0.1

the local popularity method with LFU, and for reference – a random method where resources are randomly pre-selected to be kept in every routing index. In these experiments, the workload of the resources was determined by a Zipfian with a θ coefficient of value 0.5. We pruned the routing index to 50% of its actual size.

Figure 2 shows that the LRU and random methods do not scale up as they cannot route efficiently with 50% or less resources in the routing indices. The problem arises from two causes: first, the possible irrelevance of entries in the routing indices; second, the high instability of the entries, leading to inaccuracy in the content of the routing indices. Among the methods, the local popularity method with the LFU algorithm performs well; in the experiments, its performance approached that of the global popularity strategy. Therefore, for the next experiments, we present only the results for the local popularity pruning method using the LFU algorithm.

Figures 3 to 5 show the experiment results for the local popularity pruning method using the LFU algorithm when θ was 0.9, 0.5 and 0.1 respectively. In each figure, we can see that different pruning levels do indeed produce gradual effects on routing time eventually. The experiments also show that different values of θ cause routing times to converge at different pruning levels. When we set θ to 0.9, routing times could not converge beyond the 50% pruning level. But when we set θ to 0.1, the routing times could converge even when we pruned the routing indices up to 80%. This result confirms that the routing algorithm we employ can be scaled.

Figures 6 to 8 show the experiment results from a dynamic peer-to-peer environment. To create the dynamic environment, we simulated a node joining or leaving in every t unit of time. First, we observe performance in terms of routing time from different pruning levels when θ equalled 0.5. The experiments show our approach produces good results even in a dynamic environment. We could still achieve convergence even when we pruned the routing indices by up to 40%. From the figures, we can see increases in routing time at some points, and those would have been caused by a node leaving. We can also see that soon the system tried to update its routing indices to achieve a stable state again, and that would bring routing time back to the minimum. Figure 7 clearly shows that with pruning of routing indices at 40% level, and nodes leaving or joining very 100 units of time, routing time does not fluctuate much with our method, even in a dynamic environment. We also see that the smaller the value of θ is, the better routing time will be.

Figure 8 shows our last experiment, which was to ascertain the impact of the frequency of nodes joining and leaving on routing time. The more nodes joining or leaving a network, the more dynamic the environment will be. In the experiment, we simulated a node joining or leaving every n unit of time, and we set the value of n as: 15, 25, 100, 200 and 300. The figure shows that the more dynamic the environment, the more difficult it is for the system to converge routing times. We have observed that the phenomenon is caused by instability in routing indices. When a

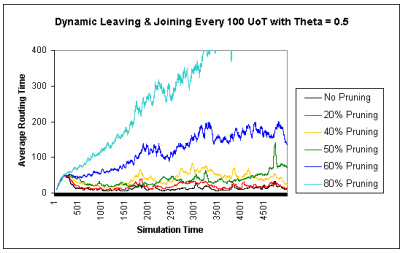

Figure 6. Dynamic leaving & joining every 100 UoT with θ = 0.5

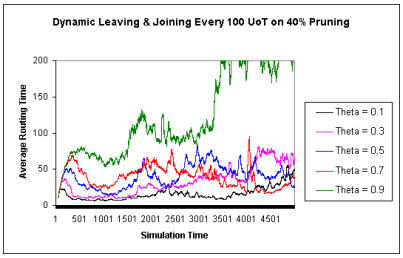

Figure 7. Dynamic leaving & joining every 100 UoT on 40% pruning

routing index is unstable, it may point to a non-optimal path or to resources belonging to nodes that have left the network. Therefore, nodes would need a longer time to search for the required resources.

6 Conclusion and Future Work

We have presented a pruning strategy for unstructured peer-to-peer systems. The strategy learns the local popularity of resources to prune the routing indices of nodes. We have shown by empirical analysis that the proposed strategy works well in both static and dynamic environments. We have also proven the viability (effectiveness, efficiency and scalability) of routing indices. In addition, we have demonstrated the adaptiveness that the reinforcement learning approach could confer.

References

[1] S. Bressan, A. N. Hidayanto, C. Y. Liau, and Z. A. Hasibuan. Adaptive double routing indices: Combining effectiveness and efficiency in p2p systems. In *Proceedings of DEXA Conference*, 2004.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Lecture Notes in Computer Science*, 2001.

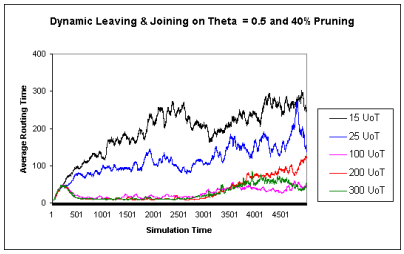

Figure 8. Dynamic leaving & joining on 40% pruning with θ = 0.5

[3] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[4] J. K. et al. Routing indices for peer-to-peer systems. In *Proceedings International Conference on Distributed Computing Systems*, July 2002.

[5] S. Kumar and R. Miikkulainen. Confidence based dual reinforcement q-routing: An adaptive online network routing algorithm.

[6] S. Kumar and R. Mikkulainen. Dual reinforcement q-routing: An on-line adaptive routing algorithm. In *Proceedings of the Artificial Neural Networks in Engineering Conference*, 1998.

[7] C. Y. Liau, A. N. Hidayanto, and S. Bressan. Adaptive peer-to-peer routing with proximity. In *Proceedings of DEXA Conference*, 2003.

[8] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications*, 1993.

[9] A. Oram, editor. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[12] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[13] Gnutella home page. In *http://gnutella.wego.com*.