

Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning

Shuming Shi, Guangwen Yang, Dingxing Wang, Jin Yu, Shaogang Qu, Ming Chen

Department of Computer Science and Technology, Tsinghua University
Beijing, P.R.China

E-mail: {ssm01, yujin, qsg02, cm01}@mails.tsinghua.edu.cn; {ygw, dxwang}@mail.tsinghua.edu.cn

Abstract—This paper discusses large scale keyword searching on top of peer-to-peer (P2P) networks. The state-of-the-art keyword searching techniques for unstructured and structured P2P systems are query flooding and inverted list intersection respectively. However, it has been demonstrated that P2P-based large scale full-text searching is not feasible by using either of the two techniques. We propose in this paper a new index partitioning and building scheme, multi-level partitioning (MLP), and discuss its implementation on top of P2P networks. MLP can dramatically reduce bisection bandwidth and end-user latency compared to the partition-by-keyword scheme. And compare to partition-by-document, it need only broadcast a query to moderate number of peers to generate precise results.

1. Introduction

As more and more data are stored in peer-to-peer (P2P) systems, keyword searching is required to help finding information more efficiently. Although dedicated search engines (like Google [5]) can be used to index P2P contents, there are potentially several advantages (e.g. data freshness, scalability, and availability) for P2P systems to support full-text search by themselves.

Two classes of techniques have been proposed to perform keyword searching in P2P networks: query flooding, and inverted list intersection. Unstructured P2P systems, such as Gnutella [4] and KaZaA [9], naturally support keyword searching by flooding queries to some or all peers. Another kind of P2P networks (structured P2P systems), such as CAN [13], Chord [17], Pastry [15], Tapestry [23], and SkipNet [12], don't support full text searching directly. While, as they actually implement distributed hash tables (DHTs) over them, keyword searching can easily be implemented by distributing inverted indices among hosts by keyword. Then a query with k keywords can be answered by at most k hosts through the intersection of inverted lists. This approach is adopted by some recent proposals [14][2][6][7] to add full text searching functionality to structured P2P systems.

However, it has been demonstrated in [10] that the above two techniques and their existing optimizations are not feasible to perform large scale keyword search. The reason is that the bandwidth consumed by them exceeds the internet's capacity. In addition to bandwidth consumption, P2P-based large scale keyword searching must have low end-user latency and high availability. Until now, it still remains a challenge to perform large scale full-text keyword searching on top of P2P networks.

From the point of view of index partitioning, the above two techniques partition indices by document and keyword respectively. Both of the two partitioning schemes have their merits and drawbacks. There also have been several hybrid index organization and partitioning schemes [1][16], but they do not directly apply to large scale, self organized, and dynamic P2P systems.

In this paper, we propose a hybrid index partitioning scheme, multi-level partitioning (MLP), which is adaptive and can achieve a good tradeoff between partition-by-keyword and partition-by-document. We describe in this paper the design of MLP on top of SkipNet [12]. MLP can dramatically reduce bisection backbone bandwidth and communication latency compare to the partition-by-keyword scheme. And, compare to partition-by-document, it need only send a query to moderate number of peers to generate precise results.

The rest of the paper is organized as follows. In section 2, we introduce briefly SkipNet and index partitioning as the background. Section 3 presents multi-level partitioning and illustrates how to implement it on top of structured P2P networks. Section 4 describes the experiment results. Related works are discussed in section 5, and we conclude in section 6.

2. Background

MLP is designed and implemented on top of SkipNet [12], of which we will first give a brief introduction. We will also briefly describe index building and partitioning techniques used in existing P2P keyword searching systems.

2.1 SkipNet

SkipNet organizes nodes into a circular distributed data structure. Any node in SkipNet has a lexicographic ID together with a numeric ID. SkipNet contains multiple levels of rings where ring members are lexicographically ordered according to node's lexicographic IDs.

One of the most important features of SkipNet is its supporting for CLB (constrained load balancing), that is, data can be uniformly distributed across a well-defined subset of the nodes in the system. The ID of any data object in SkipNet has the format of "domain!nid". The "domain" part is actually a lexicographic prefix that specifies the set of nodes over which DHT load balance should be performed. The "nid" part determines the

specific node on which the data object should be located. Please refer to [12] for more details of SkipNet.

2.2 Index building and partitioning

Inverted index is the most widely used indexing structure in full-text searching systems. It comprises of many *inverted lists*, one for each word. An inverted list for a word contains all the identifiers of documents in which the word appears.

There are two common P2P index partitioning strategies: *partition-by-document* and *partition-by-keyword*. With partition-by-document, each host maintains a local inverted index of the documents it is responsible for. Using this strategy, each query must be flooded to all peers. The partition-by-keyword strategy assigns each keyword undivided to a single node, and each node maintains the inverted lists of some keywords. For a query that contains k keywords, at most k nodes are needed to answer it, and flooding can be avoided. However, as keyword intersection requires that one or more inverted lists be sent over the network, bandwidth consumption and transmission time may both be large.

3. Multi-Level Partitioning

Multi-level partitioning (MLP) is a P2P-based hybrid partition strategy with two features: uniformity and latency locality. In this section, we describe the main idea of MLP (in section 3.1) and its implementation on top of P2P networks (in section 3.2).

3.1 Multi-level partitioning

The definition of MLP relies on a node group hierarchy. In the hierarchy, all nodes are logically divided into k groups (not necessarily to be equal size), and each group of nodes are further divided as k sub-groups. Repeat this process to obtain a node hierarchy of level l , as shown Figure 1.

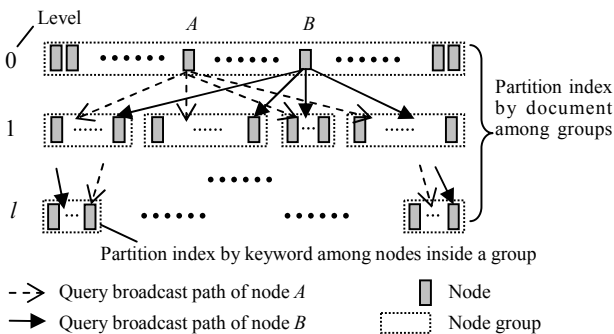


Fig.1. Node group hierarchy and multi-level partitioning

Given the hierarchical node groups, the global inverted index is partitioned by document among groups, that is, each group maintains the local inverted index for all documents belonging to this group. Now consider how to

maintain the index for a group on level l and how to process a query.

Group index maintenance For each group on level l , the index is partitioned among nodes according to the partition-by-keyword scheme, that is, each node in the group is responsible for the inverted lists of some keywords. As a result, the group index is distributed among all nodes in the group. In building index, to guarantee the balance of storage utilization, all nodes must have roughly the same number of inverted lists on them. We will illustrate, in Section 3.2, how to achieve this on top of P2P networks.

Query processing Take a query (initiated from node A) containing keyword w_1 and w_2 as an example. The query is broadcasted from node A to all groups of level 1 (see Figure 1), and then to all groups of next levels, until level l is reached. For each group in level l , the two nodes which contain the inverted lists of keyword w_1 and w_2 are responsible for answering the query. And the two inverted lists are intersected to generate the search results of the group. And then, the search results from all groups are combined level by level and sent back to node A . We can learn from the query processing process that, if all nodes have roughly the same number of inverted lists on them, then they have equal chance to answer a query request. As a result, no node will be overloaded and load balance is guaranteed.

Compared to partition-by-document, MLP avoids flooding a query to all nodes by having only a few nodes in each group to process the query. Note that the node groups from 0 to $l-1$ are virtual groups and the root group contains all nodes. So the root group should not become a single point of failure.

We call parameter l the *partition interface*. And a MLP with parameter l is called l -MLP. The partition-by-document and partition-by-keyword schemes can both be seen as special cases of MLP. With $l=0$, we get the partition-by-keyword scheme; and if l is large enough such that each group on level l contains at most one node, we get the partition-by-document scheme.

The goal of MLP is to achieve a good tradeoff among end-user latency, bandwidth consumption, load balance, availability, and scalability. To achieve this, some constraints must be satisfied:

- 1) **Uniformity** All nodes must have roughly the same number of inverted lists on them.
- 2) **Latency locality** Intra-group latency (the latency between a pair of nodes in the same group) should be smaller than inter-group latency (the latency between nodes of different groups) on each level of the node hierarchy.

Uniformity is for the balance of load and storage among peers in the system. Latency locality is for reducing end-user latency and bisection bandwidth consumption. With latency locality, all nodes in a group are roughly in the same sub-network. So, bisection backbone bandwidth is hoped to be saved by confining inverted list intersection inside each sub-network.

Now we will, in the following section, illustrate how to implement MLP on top of P2P networks to satisfy the above two conditions.

3.2 SkipNet-based multi-level partitioning

In this sub-section, we choose SkipNet as the substrate to implement MLP on top of it.

3.2.1 Lexicographic ID generation

Each node in the SkipNet has a lexicographic ID (LexID for abbreviation) and a numeric ID (abbreviated as NumID). The NumID can be generated by computing a cryptographic hash of the node’s public key or its IP address, as existing DHTs have done. The generation of LexID is not straightforward. Although [12] suggests using a node’s DNS name as its LexID, it may be not appropriate to be used here for some reasons. Firstly, some peers in the system may have not got DNS names with them. Secondly, as we will see, the LexIDs of nodes must satisfy latency locality.

We use a two-step approach to generate LexID for each node. In the first step, we model the internet as a geometric space and compute the coordinates of nodes in a d -dimensional Cartesian space, using GNP [3] or other techniques [20][11]. Then in the second step, the LexID of each node is generated by mapping from its coordinates to a circular ID space. Id-mapping can be performed using space-filling curves. Xu et al. [19] demonstrated to use space-filling curves to map points in the domain R_d into R_1 such that the closeness relationship among the points is preserved. We adopt a similar but slightly different strategy to do id-mapping. Space limitation prevents us from giving more details.

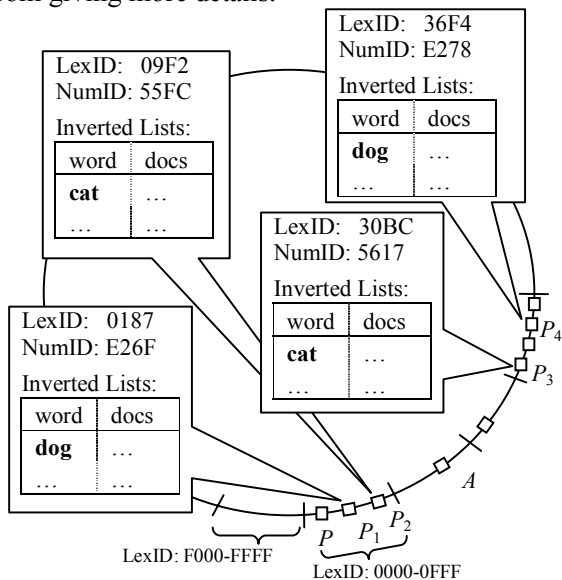


Fig.2. Index publishing on SkipNet using 1-MLP. The hash code of “dog” and “cat” are $E23D$ and $561b$ respectively. Only the

root ring of SkipNet and the level-0 of the hierarchy are displayed here.

3.2.2 Building hierarchical groups

To perform MLP, a node group hierarchy is built as follows. Firstly, all nodes are divided into groups according to the first digit of their LexIDs, as is shown in Figure 2. That is, all nodes with the same first digit in their LexIDs belong to the same group. Then, each group is divided into some sub-groups according to the second digit of LexID. Repeat this process until we reach level l . In a word, the node group hierarchy is built by dividing LexID digit by digit.

From the above process, we can easily see that latency locality is guaranteed.

3.2.3 Index building

According to the group hierarchy building process, all nodes in the same group G have a common LexID prefix, defined as $domain(G)$. To build index, each node (say node A in domain G) in the system runs the algorithm in Figure-3. In the algorithm, a node A first builds its own local index (by calling function $buildLocalIndex$) based on all the documents hosted on it. And then, for each keyword, its inverted list is published (or stored) on a destination node B in the same group. B is selected according to the group prefix and the hash code of the keyword, as follows: First hash the keyword to obtain an identifier kid . And then, the inverted list is inserted into SkipNet using “ $domain(G)!kid$ ” as the key.

```

SkipNetNode.buildIndex() {
  InvertedList[] lists = buildLocalIndex();
  for i = 1 to lists.length {
    publishInvertedList(lists[i]);
  }
}

SkipNetNode.publishInvertedList(InvertedList list) {
  domain = this.LexID.prefix(l); // l: the partition interface
  kid = list.keyword.hashcode();
  SkipNetNode node = route(domain + "!" + kid);
  store(node, list); //store list on node
}

```

Fig.3. Pseudocode for index building on SkipNet with l -MLP.

For instance, in Figure-2, consider node P and one of its keyword “cat” (whose hash code is assumed to be $561B$). As the group prefix of P is 0, the object ID for the inverted list of “cat” can be expressed as “ $0!561B$ ”. So the inverted list can be inserted into SkipNet by using this object ID as key. In Figure-2, we assume that the NumID of node P_2 is the nearest to $561B$, then the inverted list of “cat” should be stored to node P_2 .

From the above index building process, we can see that, because of the constrained load balancing (CBL) property of SkipNet, all nodes in a group have roughly the same

number of inverted lists on them. That is, the uniformity constraint in Section 3.1 is satisfied.

3.2.4 Query processing

In receiving a query request, the node runs the algorithm in Figure-4 to process it. Each query has a “level” field to indicate which hierarchical level the query is on. If the query’s level is smaller than the partition interface l , it is broadcasted further to all child subgroups along the hierarchy (see the *broadcastQuery()* function in the pseudocode). This process is repeated until level l is reached. At level l , for all keywords in the query, some query messages are sent to the nodes which contain the inverted lists for them.

```

SkipNetNode.processQuery (Query q) {
  if(q.level < l) // l: the partition interface
    broadcastQuery(q);
  else intersectQuery (q);
}

SkipNetNode.broadcastQuery(Query q) {
  q.level++;
  for s = '0' to LexID.max_digit {
    dom = this.LexID.prefix(q.level - 1) + s
    Send q to any of the nodes in domain dom
  }
}

SkipNetNode.intersectQuery(Query q) {
  domain = this.LexID.prefix(q.level);

  String words[] = q.keywords();
  for i = 1 to words.length {
    kid = words[i].hashCode();
    SkipNetNode nd = route(domain + "!" + kid);
    Send q to node nd;
  }
}

```

Fig.4. Pseudocode for query processing on SkipNet with l -MLP.

3.2.5 Discussion

From the above index building and query processing process, we can understand why MLP can achieve high performance. For multi-term queries, MLP reduces end-user latency by parallelizing the inverted list intersection process. Because of latency locality, nodes in the same group are more likely to be in the same sub-network. As a result, the intersection of inverted lists spends on average less bisection bandwidth. Compared to partition-by-document, MLP avoids flooding a query to all nodes by having only a few nodes in each group to process the query (assume there are totally M node groups on level l , then at most $2M$ nodes are bothered to answer a tow-term query).

4. Experiments

To test the feasibility and efficiency of MLP, we developed a simulated P2P searching system and done

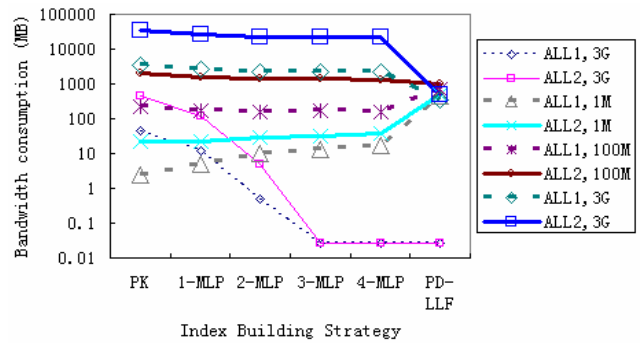


Fig.5. The aggregate bandwidth (represented by the thick lines) and bisection bandwidth (thin lines) consumption of different strategies, for various amounts of documents in the system. (16384 nodes)

some experiments based on it. In this section, we describe our experimental methodology and results briefly.

4.1 Data source and experimental methodology

We use several network configuration profiles in our experiments, all based on Transit-Stub topology [22]. All experimental results displayed in this section are generated by using a configuration profile called NET1. NET1 contains 3 transit domains and 15 stub domains, and all nodes have the upstream and downstream bandwidth of 800KB. We assign link latencies of 20ms, 5ms, and 2ms for inter-transit, stub-transit and inter-stub links respectively. We have tried other profiles, and similar results are generated.

For a network message of size s to be sent from host A to host B , the time spent is computed by the following formula (the same as in [14]):

$$Time = l(A, B) + \min(usbw(A), dsbw(B)) / s \quad (1)$$

Where $l(A, B)$ means the latency between host A and B ; and $usbw(A)$, $dsbw(B)$ represent the upstream bandwidth of A and downstream bandwidth of B respectively.

The data used for experiments are from Google programming contest data [5] which includes a selection of 900,000 HTML web pages from 100 different sites in the “edu” domain.

In experiments, we found that the performance of each partitioning scheme depends heavily on data size and query patterns. To test the effect of data size, we vary the number of documents indexed from 10^6 to $3 \cdot 10^9$ (Google index $3 \cdot 10^9$ documents [5]). Note that we have only roughly 10^6 documents in hand. We adopt *document scaling* to achieve our goal. That is, to simulate N documents, we actually build a system containing M documents and build the inverted index accordingly. While in searching, the size of inverted list for each keyword is multiplied by a ratio N/M . In the following experiments, the results for more than 10^6 documents are generated by document scaling.

To test the effect of query patterns on search efficiency, nine types of queries are generated as follows: Firstly, all

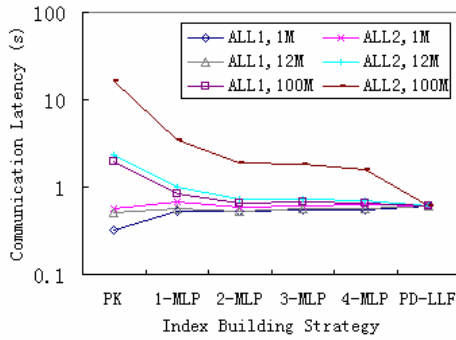


Fig.6. Communication latency of different strategies, for various amounts of documents in the system. (16384 nodes).

keywords are divided into three categories (popular, medium popular, rare) according to their appearance popularity. Three of the nine query types are one-term query types, with each query in them containing one popular, medium popular, or rare term respectively. The remaining six types are double-term queries with different combination of keyword categories. Given the nine query types, we can generate a query distribution by assigning *weights* for each query type. We use in the experiments two query distributions, called *ALL1* and *ALL2* respectively. The first distribution (*ALL1*) is generated from the Heaven Web [8] query log. The second distribution (*ALL2*) is generated by giving all the nine query types the same probability.

Due to space limitations, other experimental setup is omitted here.

4.2 Experimental results

Figure-5 shows the aggregate bandwidth and bisection bandwidth consumption. In the figure, ‘3G’ means that $3 \cdot 10^9$ documents are indexed in the system. PK and PD mean partition-by-keyword and partition-by-document respectively. We can see that, 2-MLP and 3-MLP achieve more than 80 and 1700 times of bisection bandwidth reduction respectively (although spend slightly more aggregate bandwidth when the number of documents indexed is small). This is much more than the bisection improvement needed in [10]. So, by combining MLP with the optimization techniques in [10], the bisection bandwidth is not a bottleneck any more.

Figure-6 shows the end-user latency for different data sizes. We can see that, small amounts of documents per node see the benefits of using partition-by-keyword. While for medium or large amount of documents per node, the partition-by-keyword strategy becomes unusable because of too large communication latency. Fortunately, MLP can be used to reduce communication latency effectively in this case. MLP reduces the latency for multiple popular word queries remarkably with the cost of slightly more latency for single word queries.

5. Related Work

There have been recent proposals for keyword search and its optimizations [14][2][6][7] over structured P2P networks. Reynolds and Vahdat design a distributed search engine based on DHT (distributed hash tables) and propose some boosting techniques [14]. Bhattacharjee etc. discusses how to perform result-caching on top of DHTs [2]. They use a data structure called view tree to efficiently store and retrieve prior results. These proposals are all based on the partition-by-keyword strategy, and thus share the merits and drawbacks of it. KSS (Keyword-Set Search System) in [6] partitions the index by sets of keywords to reduce intersection overhead. However, with KSS, much more storage space is needed, and the bandwidth overhead for index building may also be large.

Efficient searching techniques on unstructured P2P systems are summarized and discussed in [21]. PeerSearch [18] presents the idea of building a searching system by leveraging VSM (Vector Space Model) and LSI (Latent Semantic Indexing), based on structured P2P networks. These techniques try to improve efficiency by limiting searches to a fraction of nodes. Although improve search efficiency, they may fail to retrieve important documents.

There have been some publications [1][16] to study hybrid index partitioning among multiple machines in *parallel* or *small scale distributed systems*. Although our MLP is also ‘hybrid’, it bears several distinct features (see section 3) which make it to be used in large scale, self organized, dynamic P2P environments. In contrast, the existing hybrid index partitioning strategies do not directly apply to P2P networks.

6. Conclusions

We present in this paper a new index building strategy by leveraging multi-level partitioning on top of SkipNet, a kind of structured P2P infrastructure. It is aimed to achieve a good tradeoff between partition-by-keyword and partition-by-document.

Some optimizations, such as caching, compression, and incremental intersection, have been presented in the literature [10][14] for reducing search overhead. MLP can be combined with these techniques to further improve search efficiency. We leave this as future work.

Acknowledgments

We thank Zheng Zhang at Microsoft Research Asia for useful discussions. We thank the anonymous reviewers for their comments.

References

- [1] C. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In Proc. of the 9th String Processing and Information Retrieval Symposium (SPIRE), September 2002.

- [2] Bobby Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher and Bujor Silaghi. Efficient Peer-To-Peer Searches Using Result-Caching. IPTPS 2003.
- [3] S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In IEEE INFOCOM'02, 2002.
- [4] Gnutella. <http://gnutella.wego.com>.
- [5] Google. <http://www.google.com>.
- [6] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks. Master's thesis, Massachusetts Institute of Technology, June 2002. UCB/CSD-01-1141, UC Berkeley, Apr. 2001.
- [7] M. Harren, J. M. Hellerstein, etc. Complex Queries in DHT-based Peer-to-Peer Networks. IPTPS'02.
- [8] <http://e.pku.edu.cn>.
- [9] KaZaA. <http://kazaa.com>.
- [10] Jinyang Li, Boon Thau Loo, etc. On the Feasibility of Peer-to-Peer Web Indexing and Search. IPTPS 2003.
- [11] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Saleem Bhatti. Lighthouse for Scalable Distributed Location. IPTPS 2003.
- [12] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. USITS'03, 2003.
- [13] S. Ratnasamy, et al. A Scalable Content-Addressable Network. in ACM SIGCOMM'2001. San Diego, CA, USA.
- [14] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. Middleware'03, 2003.
- [15] Rowstron, A. and P. Druschel. Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. in IFIP/ACM Middleware. 2001. Heidelberg, Germany.
- [16] O. Sornil and E. A. Fox. Hybrid partitioned inverted indices for large-scale digital libraries. In Proceedings of the 4th International Conference of Asian Digital Libraries, Bangalore, India, 2001.
- [17] Stoica, I., et al. Chord: A scalable peer-to-peer lookup service for Internet applications. in ACM SIGCOMM. 2001. San Diego, CA, USA.
- [18] C. Tang, Z. Xu and M. Mahalingam. Peersearch: Efficient information retrieval in peer-to-peer networks. In Proceedings of HotNets-I, ACM SIGCOMM, 2002.
- [19] Zhichen Xu, Mallik Mahalingam and Magnus Karlsson. Turning Heterogeneity into an Advantage in Overlay Routing. Infocom'03, 2003.
- [20] Zhichen Xu, Chunqiang Tang, Zheng Zhang. Building Topology-Aware Overlays Using Global Soft-State. ICDCS 2003.
- [21] Beverly Yang and Hector Garcia-Molina. Efficient Search in Peer-to-peer Networks. ICDCS'02, 2002.
- [22] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In Proc. of IEEE Infocom'96, CA, May 1996.
- [23] Zhao, B. Y., Kubiawicz, J.D., and Josep, A.D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley, EECS, 2001.