A Peer-to-Peer Architecture to Enable Versatile Lookup System Design

Vivek Sawant Jasleen Kaur
University of North Carolina at Chapel Hill, Chapel Hill, NC, USA
{vivek, jasleen}@cs.unc.edu

Abstract

The resource lookup requirements in applications such as web caching, web content search, content distribution, resource sharing, network monitoring and management, and e-commerce have caught the attention of peer-to-peer (P2P) distributed systems researchers. Over the past few years, several decentralized P2P lookup system designs have been proposed for addressing these requirements. Most of these early designs are targeted at specific applications. Unfortunately, the variations in the operating environments and lookup characteristics across applications restricts the applicability of such specialized designs. In this paper, we present an architecture for P2P systems that identifies the functions necessary for designing resource lookup systems with wide applicability. We demonstrate the usefulness of the functions included in the architecture by illustrating their use in developing diverse lookup techniques.

1. Introduction

The Peer-to-Peer (P2P) service model is being intensely explored for creating scalable and robust designs for decentralized Internet-scale applications. One of the application domains receiving significant attention of P2P researchers is *resource lookup* motivated by a variety of applications including web caching, web content search, content distribution, resource sharing, network monitoring and management, and e-commerce.

The resource lookup problem involves locating resources of interest from among a large collection of resources. Designing a lookup system for these applications is made challenging by several issues including those of scale (large number of resources, participants and end hosts), volatility (high churn of participants), availability (independent or correlated failures), load balance (non-uniform resource placement and query distributions), and changing scale (sustained growth). In the past few years, numerous techniques have been proposed for addressing these challenges, and several recent designs have employed these techniques for creating decentralized P2P lookup systems for applications such as those mentioned above [1, 4, 9, 13, 14].

As is often the case, most of the early designs of lookup systems are targeted at specific applications. While these applications share most the above design issues, they also show significant differences, especially along two dimensions: operating environments and lookup characteristics. These differences limit the wider applicability of existing lookup systems designs due to one or more of the following design choices they make.

- First, a design may tightly couple application-specific functions with application-independent functions. For example, popular P2P file-sharing networks are known to be quite heterogeneous in their capacity [15]. The lookup efficiency of these networks is improved if the peers bias their neighbor selection toward high capacity peers — the high capacity nodes become the high degree nodes attracting proportionately larger query load. However, making this bias an integral part of the neighbor selection algorithm would make the system unsuitable for other applications for which the above assumptions do not hold.
- Second, the design may provide overly specialized functions or provide only a subset of functions along a design dimension restricting the applicability of the system. For example, a system may support imprecise queries efficiently but may not support precise queries as efficiently. Such undue specialization can also keep applications from evaluating alternative solutions.
- Finally, certain functions in a lookup system may lack the customizability desired by many applications. For example, the lookups in file-sharing applications may benefit from neighbor selection based on the capacity and up-times of peers, whereas the lookups in web caching applications may benefit from network-proximity based neighbor selection.

In summary, the variations in the operating environments and lookup characteristics across applications restrict the applicability of specialized lookup system designs. Furthermore, these variations can also exist within a single application; a fact that has been largely ignored so far. For example, a given application may have several different classes of queries each with significantly different query distributions. Variations can also occur over a period of time as the application grows. It is the central tenet of this paper that given the pervasiveness of the resource lookup problem it is only timely that the recent insights gained from designing several P2P lookup systems be applied toward building resource lookup system designs that support diverse requirements of applications.

In this paper, we present an architecture for P2P systems that identifies the functions necessary for designing such resource lookup systems. We also demonstrate the usefulness of these functions by illustrating their use in designing several resource lookup techniques.

2. Background and Approach

The resource lookup problem involves locating resources of interest from a large collection of resources. In the target applications mentioned earlier, the lookup problem is distributed in nature — the information about resources and the users trying to lookup those resources are distributed across the network. Such a decentralized setting is served well with the peer-to-peer service model, in which concurrently operating service points distributed across the network cooperate to implement a service. Over the past few years, several designs using the P2P model for decentralized resource lookup have been proposed. Each of these designs are called upon to address the complexity arising from the operating environment and the lookup characteristics of their target application to provide an efficient lookup solution. They achieve this by employing suitable techniques for *P2P overlay network construction* and *P2P query resolution*. In the rest of this section, we examine these techniques and their limitations to identify the requirements for a P2P resource lookup architecture with wide applicability.

2.1. P2P Overlay Construction

In the P2P model, the service points, often known as *peers*, form an application-level overlay network and communicate over this virtual network for implementing the P2P service. To participate in an overlay, each peer maintains a *neighbor table* containing information about a small set of nodes to be used for communicating with arbitrary nodes in the overlay.

The architectures for P2P overlay networks used by P2P lookup systems are divided into two categories: structured and unstructured. Structured P2P networks conform their topology to a graph structure, such as a tree or a mesh, and leverage that structure for limiting the cost of overlay network functions to a sub-linear order (e.g. $O(\log N)$) of the number of nodes in the overlay. They also provide a message routing function that maps each identifier or *key* drawn from a numeric identifier space to a unique node in the overlay, called its *authority node*, and export the abstraction of routing a message identified with a key to its authority node.

Unstructured networks, on the other hand, are ad-hoc networks. For overlay communication, They provide message dissemination abstractions, like flooding and random walk. Flooding involves each node, starting from the message source, forwarding the message to each of its overlay neighbors. A random walk involves a message being forwarded along a random path determined by each peer on the path forwarding the message to a random neighbor. A recent study implemented these abstractions on structured overlays, and proved that they can be made more efficient by leveraging the topology of structured overlays [2]. Therefore, we will consider only structured overlay architectures in the rest of the discussion.

2.2. Neighbor Selection

In structured overlays, the neighbor table is organized such that the resulting topology can be leveraged to route a message in bounded number of hops. Consequently, a given slot in the neighbor table can only be filled with a node that has an identifier from corresponding portion of the identifier space. In a network with sufficiently large number of nodes, however, multiple nodes may qualify for a given slot in the neighbor table. The simplest *neighbor selection* strategy would be to select a node at random from the candidate nodes. However, P2P applications, particularly P2P lookup, can benefit from being able to select a node based on an application-specific criteria such as node capacity or node uptime. Some of the existing designs of lookup systems support such neighbor selection. They, however, integrate the application-specific neighbor selection criterion into the basic neighbor selection algorithm, making them unsuitable for applications where other criteria are relevant. The above observation motivates the following requirement for a lookup system that is useful to diverse applications: *The P2P system should allow the application to apply an arbitrary neighbor selection criteria (within the bounds of basic structural constraints) for application-specific customization of the overlay.*

2.3. P2P Query Resolution

The lookup queries in the applications of our interest can be *precise* queries based on resource identifiers or *imprecise* ones involving keywords or attributes associated with resources, or both. For efficient resolution of queries, particularly for imprecise queries, it is often necessary to create indexes that map attribute values to resource entities (or their locations.) In the applications involving large number of resources these indexes can be quite large and are partitioned across nodes in the P2P overlay.

Index Partitioning – Attribute vs. Entity: There are two common ways of partitioning the index: attribute-partitioning (vertical partitioning) and entity-partitioning (horizontal partitioning). In an attribute-partitioned index, all the index entries corresponding to a certain attribute value are stored at the same node. To resolve a query, it is sent to the nodes storing the index entries for the attribute values appearing in the query to extract the lists of entities (the posting list) associated with those attribute values. An intersection is performed on these lists to obtain the result for the query. It is possible to achieve very high recall with this method, however, it incurs high overhead of indexing and query processing for popular attribute values [12].

In an entity-partitioned index, all the index entries for a given resource entity are stored at the same node. To resolve a query, it must be disseminated so that it can reach the nodes that have the index entries that satisfy the query. The query dissemination is commonly implemented using techniques like flooding or random walk. In this method, the index for a given entity is self-contained and can be replicated at multiple nodes to improve the performance of lookups. The entity-partitioning based systems often replicate entries in proportion with the popularity of entities to improve the average lookup performance of the system, but this does not improve the performance of less popular items.

The lookup characteristics of a given application determine which of the above schemes are suitable. For example, in an application like network monitoring, most the resources are of a local or restricted scope of interest, and hence the queries of universal interest are rare. Attribute-partitioning would work well for such applications. On the other hand, applications like file-sharing and web content search show skewed popularity distributions. Such application would find a combination of the above two schemes more effective. This motivates the following requirement from a generic lookup system that is capable of supporting diverse applications: *The P2P system should provide efficient functions to enable diverse indexing techniques—such as entity-partitioning and attribute-partitioning—used for efficient query resolution.*

2.4. Proposed Architecture

In this paper, we present a structured overlay based P2P system architecture, which identifies a set of functions necessary for addressing the above requirements. These functions are divided into two layers: overlay networking and communication functions at the lower layer and distributed state management functions at the upper layer (Figure 1.) The details of these layers are described in next two sections.

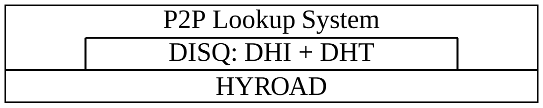

Figure 1. Layers of the Architecture

3. HYROAD

The lower layer of the proposed architecture is called HYROAD (Hybrid Routing to Arbitrary Destinations.) It identifies two categories of functions: P2P overlay networking and P2P message communication.

3.1. Overlay Networking Functions

The HYROAD networking function is primarily concerned with participation of a node in an overlay. For a particular peer, this amounts to maintaining its neighbor table with the help of its neighbors and helping them maintain theirs. The latter involves sharing the information about any known nodes in the overlay with ones neighbors. Maintaining own neighbor table involves two functions.

The first function, which we will refer to as *node information acquisition*, deals with actively acquiring or passively receiving information about the nodes in the overlay that are useful for populating the local routing table. The function uses a *join* protocol for acquiring node information when the node enters the overlay, and an *update* protocol for acquiring up to date information about the nodes of interest while the node remains a member of the overlay.

The second function, called *neighbor selection*, involves applying a selection criteria for choosing a node from a set of candidate nodes for a particular slot in the neighbor table. The HYROAD neighbor selection function provides random selection as the default selection criterion. Unlike existing P2P lookup systems, it also allows applications to apply their own neighbor selection criteria. For this purpose it defines an *up-call* interface: (i) for the system to pass the information about the candidate nodes to the application, and (ii) for the application to return its selection. This allows the application to select neighbors from the given set using arbitrary criteria, including those based on system-acquired node information like latency, node capacity, and node uptime. This relatively minor alteration to the neighbor selection function in typical structured overlays gives applications significant flexibility in customizing the overlay per their operating environment. P2P lookup systems can use this feature to customize the overlay topology to improve the efficiency of lookups.

3.2. Message Communication Functions

HYROAD defines two types of overlay communication abstractions: message routing and message dissemination.

Message Routing (KBR): The message routing abstraction defined by HYROAD is the same as the key-based routing (KBR) abstraction provided by several structured P2P systems [7]. The application addresses a message with a unique identifier, or a *key*, drawn from the identifier space used by the structured overlay, and the routing function of the system routes the message to the authority node within a bounded number of hops by using the neighbor table at each hop. P2P lookup systems can use this message routing abstraction for designing a scheme for resolving precise queries, and for resolving imprecise queries using an attribute-partitioned index.

Message Dissemination (Broadcast, Random Walk): Message dissemination abstractions are key to implementing a scheme for resolving imprecise queries using an entity-partitioned index. The message dissemination abstractions of HYROAD allow applications to propagate a message in the overlay without identifying a specific destination. HYROAD defines two such abstractions: broadcast and random walk. Broadcast involves wide propagation of a message with each node, starting from the message source, forwarding the message to each of its overlay neighbors. A random walk involves a message being propagated along a random path determined by each peer on the path forwarding the message to a random neighbor. A technique called *structured broadcast* has been proposed for various structured overlay architectures, which can disseminate a message with no or very little duplicate forwarding [2, 3, 8]. Recently, a technique for random walk over a structured overlay architecture, derived from the structured broadcast algorithm for the architecture, has also been proposed [2]. HYROAD implementations are expected to provide message dissemination abstractions by employing these kind of techniques that leverage structured overlay topology.

4. DISQ

A major challenge in designing a distributed application is that of managing the distributed state. Simple and powerful abstractions for managing distributed state go a long way in making a platform convenient for developing distributed applications. This applies to P2P lookup systems as well.

DISQ (Distributed Storage and Query), the upper-layer of the proposed architecture, identifies two abstractions for managing distributed state in structured P2P overlays: distributed hash index (DHI), and distributed hash table (DHT). These are based on popular distributed storage and lookup abstractions devised for structured overlays [7].

4.1. Distributed Hash Index (DHI)

Distributed Hash Index (DHI) uses KBR as an indexing mechanism for finding the data items stored at nodes in the overlay. It is defined in terms of the operations *insert (key, data)* and *lookup (key)*. The insert operation stores the data item at the node where the operation was invoked. Then, using KBR it forwards a message to the authority node for the key, where a *pointer* indicating the location (node) of the newly inserted data items is stored. Different instances of an data item associated with a given key can be inserted at different nodes. The lookup operation routes a message to the authority node of the given key to retrieve the pointers to all the instances of data items inserted using that key. DHI is quite similar to the distributed object location and routing (DOLR) [7, 10] but with one important difference. DOLR returns a pointer to the instance that is closest to the requesting node in terms of some distance metric like network latency. DHI removes this unnecessary restriction on the semantics of the basic lookup function, and allows the application apply its own selection criteria.

4.2. Distributed Hash Table (DHT)

DHI provides an effective indexing mechanism for the distributed state stored at various nodes in the overlay using insert operation. However, when a node leaves the network, the state stored on that node becomes inaccessible. An application may desire its distributed state to be relatively persistent even as nodes join and leave the network. The popular abstraction of distributed hash tables (DHT) addresses this requirement [6, 7]. DHT is defined in terms of the operations *put (key, data)* and *get (key)*. The put operation stores the data item at the authority node for the specified key by routing a message to that node using KBR. The get operation retrieves the data item corresponding to the given key by similarly routing a message to the authority node of the key. This allows the data to be accessible even after its publisher leaves the network. To keep a data item available even after its authority node leaves the network, it is replicated on a set of nodes that are likely to become the authority node for that item if its current authority node leaves the overlay.

5. Designing with DISQ/HYROAD

The main objective of the architecture presented in the previous sections is to identify functions that can enable us to design P2P lookup systems with wide applicability. In this section, we will discuss how a system based on the proposed architecture can support diverse lookup techniques, currently available in separate designs. We will also discuss some new techniques enabled by the architecture.

5.1. Overlay Construction

As noted earlier, operating environments vary across the target applications of P2P lookup systems. Moreover, there is often significant heterogeneity within the operating environment of even a single application with respect to parameters such as inter-node latency, node capacities, and node up-times for the participants of an overlay [15]. It has been shown that adjusting the structure of the overlay in accordance with the characteristics of the operating environment can help in making lookups more efficient [2, 4]. The application-specific neighbor selection of HYROAD enables precisely this kind of customization on an application-specific basis. For example, the lookups in file-sharing applications may benefit from neighbor selection based on the capacity and up-times of peers, whereas the lookups in web caching applications may benefit from network proximity based neighbor selection. But the criteria for custom neighbor selection need not only be based on such information made available by the underlying P2P system. HYROAD also allows the use of criteria involving application-level information. We discuss a lookup technique based on one such criteria below.

Improving Lookups with Semantic Association: The analysis of query workloads of popular P2P resource lookup applications has revealed that the interest of two or more peers in the same set of entities can be used as a predictor of their future shared interests, particularly in less popular entities. Studies have reported the benefits of using the knowledge of such *semantic association* between peers for identifying peers that are more likely to resolve a given query [5, 11, 16]. A P2P lookup system based on HYROAD can use semantic association as a criterion for neighbor selection, to improve entity-partitioned index based lookups.

The real challenge in using semantic associations is identifying such associations. The techniques proposed in the literature identify the semantic associations in one of the two ways. In the first approach, when a query from peer A is resolved by peer B, both A and B identify a semantic association with each other. More such resolutions between the pair indicates greater degree of association between the two. In the other approach, if peer C resolves a query for an entity E from peer A and peer B, it notes that as semantic interest between A and B informs them about it. In both cases, A and B (and, in the second case, C) can use these associations while routing future lookup queries.

The above techniques measure the shared interest relatively locally, and hence, may underestimate the degree of semantic association or miss certain associations. For example, in the first approach, if the query from C for an entity is resolved by D, A and B will not discover about the interest they share with C and D. Similarly, in the second approach, C would not discover its shared interest with other peers that resolve the queries for E. A P2P lookup system based on the proposed architecture can broaden the scope of semantic association identification using a simple technique based on the DHT abstraction. The interest in a given resource entity E can be tracked by creating a DHT record. Whenever a query result contains E, its DHT record would be updated to note the peers originating and resolving the query. Each peer would maintain a list of its semantic peers. For a successfully resolved query, the peers involved in the query would examine the DHT record for each resource entity appearing in the query result to see if they need to update their list of semantic peers. A simple criteria based on counting the number of records in which a certain peer appears can be used to control the list membership.

5.2. Query Processing

Supporting precise and imprecise queries broadens the applicability of a P2P lookup system. Both DHI and DHT abstractions enable a lookup system to directly provide support for precise queries, as the underlying KBR mapping effectively serves as an index of resource identifiers. For imprecise queries, an explicit index that maps attribute values to resource location must be maintained across the nodes in the overlay. Both ways of maintaining such decentralized index, attribute-partitioning and entity-partitioning, can be supported by a P2P system based on the proposed architecture as described below.

P2P Lookups with Attribute-Partitioned and Entity-Partitioned Indices: In an attribute-partitioned index, all the index entries corresponding to a certain attribute value are stored at the same node. A P2P system can use the DHT abstraction for implementing an attribute-partitioned index as follows. Each attribute value is assigned a KBR key. The index entries corresponding to a given attribute can then be stored using the DHT. To resolve an imprecise queries, all the attribute values appearing in the query will be looked up in the DHT to obtained their associated index entries, which are basically lists of resources. An intersection (or appropriate join) on these lists will produce the result for the original query. Several existing designs have variations of this technique for supporting query resolution based on an attribute-partitioned index [1, 9, 13, 14]. In an entity-partitioned index, all the index entries for a given resource entity are stored at the same node. To resolve a query, it must be disseminated so that it can reach the nodes that have the index entries that satisfy the query. A

P2P system based on the proposed architecture can support query resolution based on an entity-partitioned index by using a combination of functions: the DHI abstraction for maintaining entity-partitioned index and the message dissemination abstractions of HYROAD for resolving imprecise queries using that index.

Caching Imprecise Queries A lookup system can further improve the lookup performance of imprecise queries by using the following technique that is independent of the type of index partitioning used. The system would map a given successful imprecise query to a unique KBR identifier, and *cache* the result obtained for the query into the DHT by using that identifier. While resolving an imprecise queries, it will use the same mapping scheme, and use the identifier to first lookup the DHT to check if the query is cached there. If not, it will fall back to resolving it using one of the above two methods. Unlike precise queries, imprecise queries and the resource entities corresponding to those queries do not have a one-to-one mapping. However, as evident from the lists of popular keywords published by web search engines, certain imprecise queries do appear more frequently than others. These can serve as effective cache identifiers.

6. Conclusion

The resource lookup requirements in applications such as web caching, web content search, content distribution, resource sharing, network monitoring and management, and e-commerce have caught the attention of peer-to-peer (P2P) distributed systems researchers. Over the past few years, several decentralized P2P lookup system designs for addressing these requirements have been proposed. As is often the case, most of these early designs are targeted at specific applications. Unfortunately, the variations in the operating environments and lookup characteristics across applications limits the applicability of such specialized designs.

In this paper, we present an architecture for P2P systems that identifies the functions necessary for designing resource lookup systems with wide applicability. In selecting and defining these functions, we attempt to avoid two pitfalls restricting the applicability of existing systems: undue coupling between application-specific and application-independent functions, and hardcoding application-specific optimizations into the design. The resulting architecture allows key functions like neighbor selection to be customized, and enables diverse lookup techniques to be implemented within the same system. We demonstrate the utility of the functions included in the architecture by illustrating their use in developing diverse lookup techniques. Supporting a range of techniques not only makes such systems broadly applicable, but also makes it possible for their applications to evaluate alternative techniques with relative ease.

Finally, it is worth noting that the customizability of HYROAD neighbor selection function and DHI lookup function is useful to other P2P systems besides P2P lookup, such as end-system multicast and content distribution networks.

References

[1] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architect ure for intentional resource discovery. In *Pervasive 2002*, Aug 2002.

[2] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of NSDI'05*, May 2005.

[3] M. Castro, M. B. Jones, A.-M. Kermarrec, A. I. T. Rowstron, M. Theimer, H. J. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proceedings of the 22th Infocom*, Mar 2003.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of SIGCOMM 2003*, Aug 2003.

[5] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of the 22th Infocom*, Mar 2003.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the Eighteenth ACM symposium on Operating systems principles (SOSP)*, Oct 2001.

[7] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In *Proceesings of IPTPS'03*, Feb 2003.

[8] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured P2P networks. In *Proceesings of IPTPS'03*, Feb 2003.

[9] L. Garces-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. Data indexing in peer-to-peer DHT networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

[10] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Aug 2002.

[11] A. Iamnitchi and I. Foster. Small-world fi le-sharing communities. In *Proceedings of the 23rd Infocom*, Mar 2004.

[12] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proceesings of IPTPS'03*, Feb 2003.

[13] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing P2P fi le-sharing with an internet-scale query processor. In *VLDB*, 2004.

[14] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on planetlab with SWORD. In *WORLDS '04*, Dec 2004.

[15] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer fi le sharing systems. In *Multimedia Computing and Networking (MMCN'02)*, Jan 2002.

[16] K. Sripanidkulchai, B. M. Maggs, and H. Zhang. Effi cient content location using interest-based locality in peer-to-peer systems. In *Proceedings of the 22th Infocom*, Mar 2003.