An Efficient Clustered Architecture for P2P Networks

Juan Li Son Vuong
Computer Science Department, University of British Columbia
{juanli, vuong}@cs.ubc.ca

Abstract

Peer-to-peer (P2P) computing offers many attractive features, such as self-organization, load-balancing, availability, fault tolerance, and anonymity. However, it also faces some serious challenges. In this paper, we propose an Efficient Clustered Super-Peer P2P architecture (ECSP) to overcome the scalability and efficiency problems of existing unstructured P2P system. With ECSP, peers are grouped into clusters according to their topological proximity, and super-peers are selected from regular peers to act as cluster leaders and service providers. These super-peers are also connected to each other, forming a backbone overlay network operating as a distinct, yet integrated, application. To maintain the dynamically adaptive overlay network and to manage the routing on it, we propose an application level broadcasting protocol: Efa. Applying only a small amount of information about the topology of a network, Efa is as simple as flooding, a conventional method used in unstructured P2P systems. By eliminating many duplicated messages, Efa is much more efficient and scalable than flooding, and furthermore, it is completely decentralized and self-organized. Our experimental results prove that ESCP architecture, combined with the super-peer backbone protocol, can generate impressive levels of performance and scalability.

1. Introduction

In very large networks, it is not always easy to find desired resources. For any given system, the efficiency of any search technique depends on the needs of the application. Currently, there are two types of P2P lookup services widely used for decentralized P2P systems [2]: structured searching mechanism and unstructured searching mechanism.

Structured systems such as Tapestry [6], Pastry [4], Chord [5], and CAN [3] are designed for applications running on well-organized networks, where availability and persistence can be guaranteed. In such systems, queries follow well-defined paths from a querying node to a destination node that holds the index entries pertaining to the query. These systems are scalable and efficient, and they guarantee that content can be located within a bounded number of hops. To achieve this performance level, the systems have to control data placement and topology tightly within their networks. However, this results in several limitations: first, they require stringent care in data placement and the deployment of network topology. Thus, the methods they use are not applicable to the typical Internet environment, where users are widely distributed and come from non-cooperating organizations. Second, these systems can only support search-by-identifiers and lack the flexibility of keyword searching, a useful operation for finding content without knowing the exact name of the object sought. Third, these systems offer only file level sharing, and do not share particular data from within the files.

Unstructured systems like Gnutella [1] and FastTrack [7] are designed more specifically for the heterogeneous Internet environment, where the nodes' persistence and availability are not guaranteed. Under these conditions, it is impossible to control data placement and to maintain strict constraints on network topology, as structured applications require. Currently, these systems are widely deployed in real life.

The present paper focuses on building a P2P lookup application for integration into arbitrary dynamic networks that cannot be controlled. We thus concentrate on unstructured P2P systems, which support many desirable properties such as simplicity, robustness, low requirement for network topology and supporting keyword searching. Unstructured systems operate under a different set of constraints than those faced by techniques developed for structured systems. In unstructured systems, a query is answered by flooding the entire network and searching every node. Flooding on every request is clearly not scalable, and it has to be curtailed at some point, therefore it may fail to find content that is actually in the system. Furthermore, a network that uses flooding might be bombarded with excess messages and activity, and at certain points it might fail. To address these problems, we propose a hierarchical structure and an efficient routing strategy.

2. Multi-tier architecture

In a network, participating peers exhibit considerable heterogeneity in terms of storage capacity, processing power, bandwidth and online availability. For the best design, we should take advantage of this heterogeneity and assign greater responsibility to the peers that are capable of handling it. ECSP utilizes these differences in a hierarchical P2P design, in which peers with different capabilities take different roles. Specifically, peers in the system act as client peers and super-peers in different hierarchies.

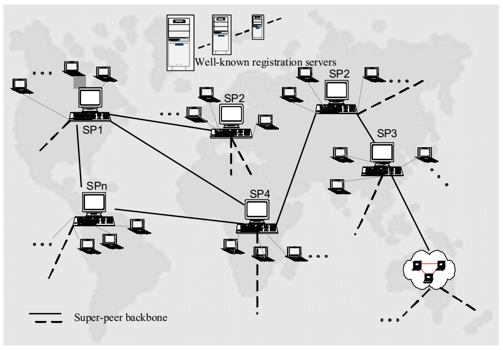

Figure 1: System architecture

Figure 1 illustrates the hierarchical structure, in which peers are grouped together if they are topologically close. Peers with more *resources* in the cluster can be selected as a super-peer. Super-peers act as local search hubs, building indices of the content files shared by each peer connected to them, and proxying search requests on behalf of these peers. Desirable properties for super-peers include accessibility to other peers, bandwidth and processing capacity. Super-peers with these characteristics are connected with each other and organized amongst themselves into a backbone overlay network on the *super-peer tier*. Then, an application level broadcasting protocol is designed to perform distributed lookup services on top of this overlay network. A unique well-known registration server is responsible for maintaining user registrations, logging users into the system, and bootstrapping the peer discovery process.

The hierarchical structure of this system combines advantages of both centralized and pure P2P systems. The introduction of a new level of hierarchy in the system increases the scale and speed of query lookup and forwarding processes. Moreover, the hierarchical structure is more stable because clusters join and leave the network less frequently than individual peers. Finally, our super-peer overlay routing protocol reduces the workload of super-peers significantly by avoiding many flooding duplications.

2.1. Well-known server

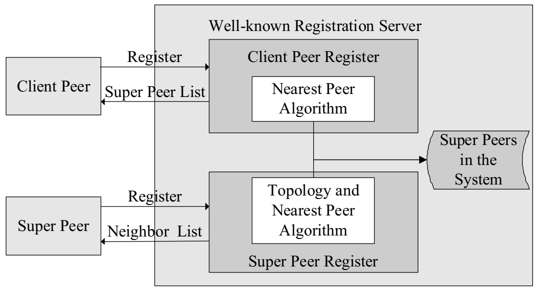

Figure 2 Well-known registration server structure

In the networks analyzed for the current study, well-known registration servers (Figure 2) supply yellow page services to all nodes in a network. Registration servers maintain databases of all active super-peers in the system, and when a new super-peer is added to the network, a new entry is generated in the registration server's super-peer database. Whenever a new peer joins the system, it first contacts the registration server to get a super-peer list. To provide scalability and load balancing, some hierarchical registration servers are essential, which contain replicas of the active registration server. Replica registration servers become active only when the main registration server is not able to provide service to nodes in the system, for example during busy periods or failing times.

2.2. Super-peers

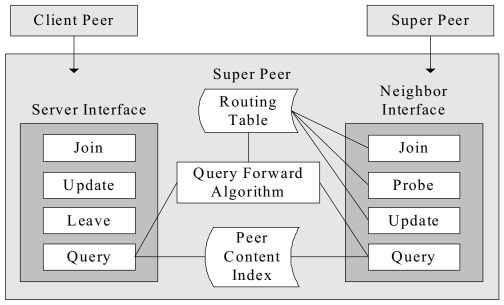

Figure Super-peer structure

Super-peers are selected from regular peers according to their computing resources and bandwidth capabilities, the volume of files they store, and the behavior of being seldom offline. Super-peers act as cluster leaders and service providers for a subset of client peers, providing four basic services to the clients: join, update, leave and query.

In the join process, client peers upload metadata describing the property of the content they will share with the network. In addition, the super-peer also stores details related to the client peer's connection, such as the IP address, bandwidth, and processing power of the client. After the join process is completed, the client peer is ready to query content in the network, and to allow other client peers to download content from it. When a client peer leaves the system, the super-peer removes that client peer's metadata from the index library. If a client peer ever updates its content data, it sends an update message to the super-peer, and the super-peer updates its index accordingly. When a super-peer receives a query from its client peer, it matches what is in its index library and forwards the query to its neighbors, who in turn forward it to some of their neighbors, according to the super-peer overlay network routing algorithm Efa. After results (or time-outs and error messages) are received from all of its neighbors, the super-peer sends the aggregated result to the requesting client peer.

As mentioned, super-peers are also connected with each other to form an application-level overlay network. The dynamic maintenance of the topology, and the efficient locating of content within this overlay network is described in the next section. Here, we note that super-peers are not only cluster leaders for their client peers, but also members of the super-peer overlay network. Therefore they supply interfaces to both client peers and to adjacent neighbor super-peers (Figure 3).

2.3. Client-peers

In the present paper, regular peers are referred to as *client* peers to distinguish them from super-peers. In fact, they act as both clients and servers: they send requests to super-peers like clients, and receive other peers' file download requests like servers. While providing this functionality, client peers can offer easy-to-use interfaces, through which users can connect to the system, discover resources in the network and finally obtain the required content. To accomplish this, a client peer acts as both an FTP client and an FTP server. After the client peer joins the system and uploads its content metadata to its local super-peer, it initiates an FTP server on a well-known port and waits for other peers' download requests. After a client peer locates content through super-peers, it opens a connection and downloads directly from the node where the content is located.

2.4. Backup peers

The introduction of one more level of hierarchy makes the system more efficient, but the super-peer becomes a potential area of single-point failure for its cluster. When the super-peer fails or leaves the system, the entire cluster content index information is lost. To increase the reliability of the system, we introduce a backup peer as redundancy for the super-peer. Thus, every cluster has a super-peer acting as a cluster leader and a backup peer acting as a redundancy server. The backup peers are selected from the client peers too. They copy the super-peer's index table periodically, and when a super-peer fails or leaves the network, its backup peer replaces it and the cluster selects a new backup peer for redundancy. The possibility of both a super-peer and its backup peer failing simultaneously is much smaller than failure of the super-peer alone, and thus the introduction of a backup peer greatly improves a system's robustness. Furthermore, a backup peer is dynamically selected from client peers in the cluster, so there is no extra burden for the redundancy.

3. Backbone overlay routing

3.1. Algorithm description

Our algorithm aims at suppressing flooding by reducing the number of duplicated query messages. There are many approaches to eliminating flooding, the most popular of which uses tree-based broadcasting. In our model, the number of participant nodes can be quite large and users are widely distributed all over the Internet. Therefore it is impossible to let every node know the whole topology of the network. In addition, all tree-based approaches require huge messaging overhead, associated with construction and maintenance of the spanning tree. However, in most P2P systems, participant nodes are typically PCs at homes or offices with their own tasks, and thus they cannot afford many resources for P2P applications. In addition, they can be very dynamic, so messages updating tree structures overwhelm the network. In light of these considerations, our objective is to use limited topology information and simple computing to decrease the duplication queries created by flooding.

In a well-connected network, several different paths may exist to connect two particular nodes, which is the reason that extensive duplications may be created by flooding. If node v can anticipate that one of its neighbors u, receives query messages from another path, however, then v does not forward the query to u. To achieve this type of anticipating, we use a rule directing the nodes that duplicate and forward messages while we keep track of topology information to compute the forwarding set. As the later experiment shows, although we cannot avoid all duplications, we can reduce much duplication for most widely used network topologies.

The following definitions are used in the algorithm and discussed later in this chapter.

- *id(v)*: Node *v*'s unique id.
- *N(v)*: Neighbor set of .*v*
- *NN(v)*: Neighbor's neighbor set of .*v*
- *fr(u,v)*: *v* is the current node, *u* is the node which forwards the query to *v*. *fr(u,v)* is the forward reaching set of *u* for the current node *v*, that is, the immediate (no more than 2 hops away) set of nodes reached by the local flooding source *u*.
- *routing(u,v)*: For local source *u*, current node *v*'s routing set. For example, if *u* forwards the query package to *v*, the set of nodes *v* forwards is decided by *routing(u, v)*.

The algorithm in figure 4 describes the routing process for the current node, *v*, when it receives a query from its neighbor, *u*.

```
forward(u,v)
/*when node v receives forwarded query from its neighbor
u, this algorithm decides how v forwards this query */

If  the received query has been received before
        discard it
else
    if  u is null /* v is the node which initiates the query*/
        forward the query to N(v)
    else
        forward the query to routing(u,v)
```

(a)

$fr(u,v) = N(u) \cup \{$ all v' in $NN(u) \mid id(v') < id(v)\}$

$routing(u,v) =$ all v' in $N(v)$, such that

1. $v' \notin fr(u,v)$ AND
2. $\{N(v') \cap fr(u,v) = \varnothing\}$ OR $\{N(v') \cap fr(u,v) = A$ AND $(\forall v'' \in A$ AND $id(v'') > id(v))\}$

(b)

Figure 4: Routing algorithm, where (a) is the routing process, and (b) is the algorithm to compute fr(u,v) and routing(u,v).

We use an example to explain the algorithm. Figure 5 depicts a simple network topology, where N5 is the current node. In the case of flooding, when *N5* receives message sent from *N1*, *N5* would forward the message to all of its neighbors except *N1*. Therefore, *N5* forwards the messages to *N3, N6*, and *N8*. However, if it uses Efa, *N5* does not need to forward the message to all of its neighbors, but only to those that may not be reached by *N1*. Messages from *N1* reach all neighbors of *N1*, which are *N2* and *N7*. Because *id(N2)* is smaller than *id(N5)*, the message also reaches *N2*'s neighbors, *N3* and *N4*. Finally, we get: *fr(N1,N5)={N2, N7, N3, N4}*. Therefore, when *N5* receives a message from *N1*, it does not forward the message to its neighbor *N3*, because *N3* is in the set *fr(N1, N5)*. *N5* does not forward the message to *N6* either, because *N6*'s neighbor *N4* is in *fr (N1, N5)*, and *id (N4)* is smaller than *id (N5)*. So at last, *N5* only forwards the message to *N8*.

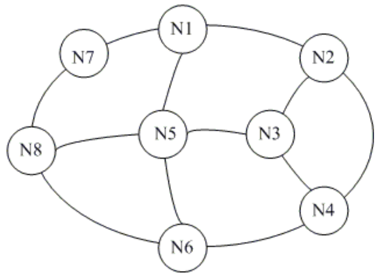

Figure 5: A simple network topology

3.2. Algorithm correctness

Assuming the network is connected, the protocol described above guarantees a query message be forwarded to all nodes in the network. For an arbitrary node *v*, which receives forwarding query from its neighbor node *u*, the entry *(u,v)* in the routing table of *v* decides which neighbors the query would be forwarded. And the entries in the routing table is computed with the following principle: If *v*'s neighbor $x \in fr(u, v)$, then *v* need not forward the query to *x*, because *v* knows *x* has been reached by *u*. If $x \notin fr(u)$, but *x*'s neighbor $y \in fr(u)$, we compare the *id* of *y* and *v*, if *v*'s id is smaller, then *v* forward the query to *u*, otherwise, v leave the query for *y* that has a smaller *id* to forward the query. Therefore, for an arbitrary neighbor of *v*, it would be reached either by *v* or by other nodes in *u*'s reaching set, whose *id* is smaller than *v*'s. Consequently, all nodes in the network will be reached by the forwarding protocol.

4. Experiments

We have performed two kinds of experiments to evaluate the system. First, we designed a simulator to evaluate the performance and scalability of the routing protocol, because it is impossible to run the system in an Internet-based network with millions of computers. Second, we installed our P2P software on the LAN of the Computer Science Department of UBC to test and evaluate the real system.

4.1. Routing protocol test

In order to evaluate the super-peer level overlay network's discovery mechanisms, we have developed an event-based simulator in Java. The simulator can simulate the application-level broadcasting and query searching processes with different routing algorithms and network topologies.

Minimizing the system overhead is an important objective for our algorithm. In our experiments, we define the overhead as the duplicated messages on the network. Figures 6 compare the system overhead when using Efa with the overhead when using simple flooding. Our experiments run on three different network topologies: grid topology, random topology and Barábasi-Albert random topology [8]. In the simulation, we set the TTL to "unlimited," to make the broadcast reach every node in the network. For each topology, we vary the network size and repeat the tests ten times, then compute the average results. The results reveal that Efa greatly reduces network overhead for all three topologies, compared with flooding.

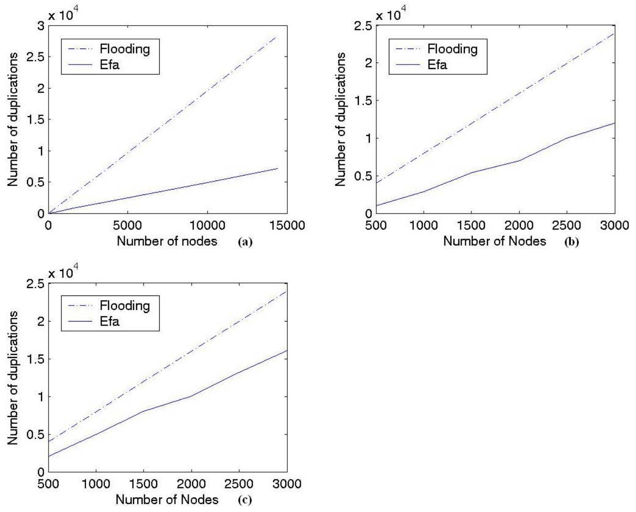

Figure 6: System overhead vs. network size for different network topology: (a): grid topology, (b): random topology, (c): Barábasi-Albert random topology

Figure 7 depicts the relationship of network duplication ratios and network average degrees. The experiment is performed on a random topology network with 3000 nodes. The network duplications increase with the network average degree in the flooding situation. For Efa, when network average degree grows to some extent, the duplication ratio begins to decrease with the increase of average node degrees.

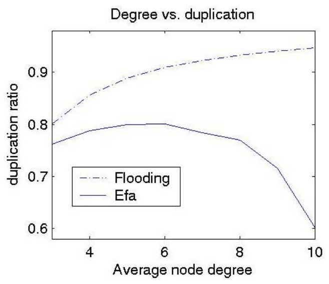

Figure 7 Degree vs. duplication

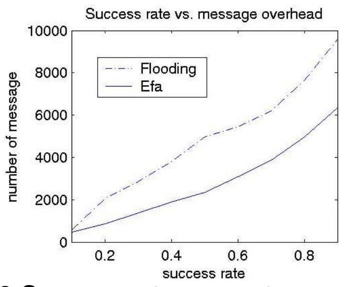

Figure 8 Success rate vs. system overhead

The experiment in Figure 8 is performed on the random topology with 3000 nodes and an average degree of 5. The content is replicated at 0.3% of the randomly selected nodes in the network. The result in Figure 9 identifies the relationship of query success probability to the number of messages produced in the system.

All experiments performed on different network environments demonstrate that compared with simple flooding, Efa reduces many overheads of individual nodes as well as the loads of the whole network. It achieves better performance and scalability than flooding does, especially when the network is well connected or the network size is large.

4.2. Real system test

The experiment environment is made up of 16 PCs with Intel Pentium Ⅲ 1.004 GHz processor and 256M of RAM, and all the PCs are running the Red Hat Linux 9 operating system. There are a total of 50 different files in the system. Every peer maintains 20 files and each of the files is around 5KB. To test our architecture, we randomly choose one to serve as a well-known registration server and the other 15 PCs to serve as peers. The 15 peers are grouped into three clusters. In every cluster, a peer also acts as a super-peer. To evaluate the system performance, we compare it with a Gnutella system. The topology of Gnutella is randomly generated with an average degree of 4. In both systems, to generate the network traffic peers send queries every two seconds.

Because the experiments are conducted on a LAN, the transmission time between two nodes is too short to reflect the real Internet environment, therefore we add 0.1 second delay for every transition between two nodes.

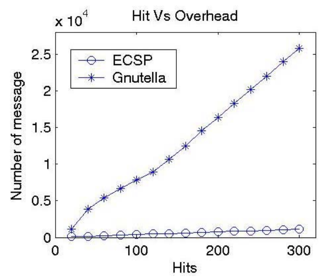

Figure 9 Hit rates vs. system overhead

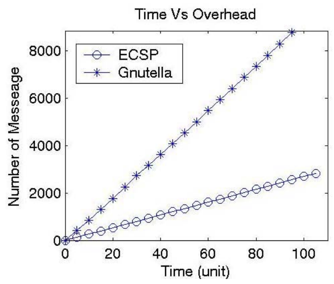

Figure 10 Time vs. system overhead

Figure 9 shows the query hits and number of messages needed. To attain the same number of successful query hits, ECSP sends significantly fewer messages than Gnutella does.

Figure 10 reviews the relationship of time consumed and system overhead: for any time period, our system creates less traffic than Gnutella does. Therefore, our system accrues lower costs than Gnutella.

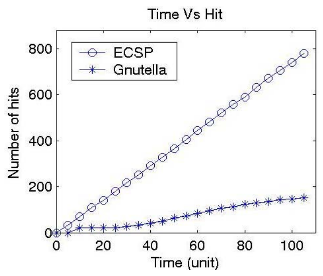

Figure 11 Time vs. query hits

Figures 11 and 12 compare ECSP and Gnutella in terms of query hits and completion time. Two observations can be drawn from these comparisons: as Figure 12 shows, our system uses much less time to finish the same amount of queries; with the same time limit, our system can finish more queries (Figure 11).

In sum, all of our experiments prove that the ECSP structure and the Efa backbone routing protocol dramatically decrease the cost of queries without decreasing the ability to satisfy queries, compared with Gnutella and simple flooding.

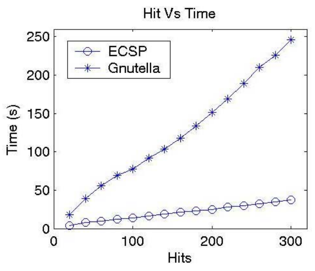

Figure 12 Query hits vs. completion time

5. Conclusions

In this paper, we investigate P2P systems currently in use, primarily on decentralized, unstructured systems. Two major deficiencies of unstructured P2P networks are addressed: scalability and efficient search mechanisms. Consequent to our observations, we propose a hierarchical-based super-peer structure, ECSP. Experiments are performed both with a real network environment and with simulation tools. The experimental results demonstrate that the ESCP architecture and the overlay broadcasting algorithm achieve good performance and scalability, and they can be used to construct powerful infrastructures for very large scale, unstructured P2P environments.

.

6. References

[1]] Gnutella website. http://gnutella.wego.com/
[2] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker "Search and Replication in Unstructured Peer-to-Peer Networks," in Sigmetrics, June 2002.
[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network:, In ACM SIGCOMM, pages161-172, August 2001.
[4] A. Rowstron and P. Druschel. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), November.
[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H.Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," In ACM SIGCOMM, pages 149-160,August 2001.
[6] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Technical Report UCB/CSD-01-1141, April 2000.
[7] FastTrack website http://www.fasttrack.nu/
[8] A.L. Barábasi and R. Albert, "Emergence of Scaling in Random Networks". Science, pages 509-512, October 1999.