Early Measurements of a Cluster-based Architecture for P2P Systems

Balachander Krishnamurthy, Jia Wang, Yinglian Xie

I. Introduction

Peer-to-peer applications such as Napster [4], Freenet [1], and Gnutella [2], [7] have gained much attention recently. These applications are mainly designed and used for large-scale sharing of MP3 files. In such systems, end-hosts self-organize into an overlay network and share content with each other. Compared to the traditional client-server model, files are served in a distributed manner and replicated among the network on demand. Since hosts participating in peer-to-peer (P2P) networks also devote some computing resources, such systems scale with the number of hosts in terms of hardware, bandwidth, and disk space. With the wide deployment of P2P applications, the P2P traffic is becoming a growing portion of the Internet traffic. There has been very little examination of P2P traffic patterns and how they differ from traditional service models. Studying and understanding P2P traffic is thus important to provide efficient application-level content location and routing within the network.

The existing applications use their own approach to do content location and routing and none of them are scalable. Napster uses a centralized server to locate content, while Gnutella clients broadcast queries to all their neighbors. [8] discusses the query locality observed in Gnutella traces and suggests caching as a short-term approach to increase Gnutella's scalability. Recent designs such as CAN [5], Chord [9], Pastry [6], and Tapestry [10] propose distributed indexing schemes based on hashing to locate content. These systems assume a flat content delivery mesh. Each object's location is stored at one or more nodes selected deterministically by a uniform hash function; queries for the object will be routed incrementally to the node. Although hash functions can help locate content deterministically, they lack the flexibility of keyword searching—a useful operation to find content without prior knowledge of exact object names. There is no real deployment at present and thus no measurement information is available for understanding the usability and scalability of such systems.

We present some early measurements of a Cluster-based Architecture for P2P (CAP) systems—a decentralized, peer-to-peer content location and sharing system that uses network-aware clustering [3]. Network-aware clustering is an effective technique to group clients that are topologically close and under common administrative control. The introduction of one more hierarchy is aimed at scaling up query lookup and forwarding. CAP would also be more stable since clusters join and leave the network less frequently than individual clients. CAP also does not use hash functions to map objects to locations deterministically. Instead, CAP behaves more like a distributed searching system such as Gnutella. We modified Gnutella and collected P2P traces from a variety of places. We analyze the trace data using network-aware clustering. We use the Gnutella trace in our simulations to measure and compare the performance of CAP and Gnutella. The implementation of CAP is currently ongoing and we plan to measure the system based on real deployment. Our trace analysis and preliminary simulation results show that the P2P network can be very dynamic, and CAP is promising in increasing the stability and scalability of such distributed applications. We are now carrying out a longer and broader study.

II. CAP

CAP employs network-aware clustering technique for content location and routing. A user query consists of names of the files to be retrieved or keywords to be searched. The query response is a tuple: (*timestamp*, *query*, *object*, *location*). Given a user query, CAP locates nearby copies of object that satisfy the query. The actual data retrieval will be performed by the user who initiated the query.

CAP uses a centralized server (called *clustering server*) to perform network-aware clustering and cluster registration. Based on the information provided by the clustering server, users are grouped into clusters and self-organize into an overlay network. The two basic operations performed in CAP are: *node joining and leaving*, and *query lookup and routing*. Users (also called *nodes*) can join and leave the overlay network dynamically. A query for an

The authors are with AT&T Labs–Research, Florham Park, NJ, USA. email:{bala,jiawang,ylxie}@research.att.com. Contact author: Balachander Krishnamurthy, Fax: 973-360-8077, 180 Park Avenue, Florham Park, NJ 07932. Yinglian Xie is a student at CMU interning at AT&T Labs–Research

object will be routed through the overlay network until locations of the desired object are found, and the location information is returned directly to the initiating node.

To help distributed query lookup and forwarding, there are one or more *delegate nodes* within each cluster. The clustering server keeps track of the existing clusters and the corresponding delegate node's information. The delegate node acts as a directory server for the objects stored at nodes within the same cluster. Queries will be submitted to the cluster delegate node first; if it cannot be resolved at the delegate directory, the delegate node forwards the query to other nodes in either a recursive or iterative way until an answer is found. To reduce query latency, delegate nodes maintain a cache of recently received queries, with each entry consisting of a query and the corresponding response. Thus the location information of popular objects will be replicated at multiple places and can be found quickly. The delegate node also performs node membership registration. It maintains information about each active node within a cluster to distinguish queries from inside and outside a cluster. In case of delegate node failure, each node independently resorts to the clustering server for the new delegate node. The first node asking for the clustering server will become the newly selected delegate node. A bootstrap mechanism is required to set up the directories at the new delegate node.

III. EXPERIMENTS

We modified an open source Gnutella client [7] to passively monitor and log all Gnutella messages that were routed through it. The end host running the modified Gnutella client joins Gnutella using Clip2's *gnutellahosts.com* [2] service. Each entry in the trace has the following fields: (1) Time stamp. (2) IP address of the neighbor host. (3) Message ID. (4) Type of message. There are four types of messages recorded: ping request (Gnutella init message), ping reply (Gnutella init response), search request, and search reply. In addition to the above fields, we also recorded other fields, which are specific to different types of messages, including query strings, number of results found, file names of returned results, etc. Five traces were collected independently at CMU(Pennsylvania), AT&T(New Jersey), ACIRI(California), WPI(Massachusetts), and UKY(Kentucky). The data gathering in the first three sites ran with unlimited number of concurrent connections. When an intrusion detection system was triggered (incorrectly), we rate-limited our experiments (reducing number of concurrent connections and the number of hosts contacted) and ran it on two other sites. The average number of neighbors of the Gnutella client in the trace can be controlled by the number of concurrent connections, with up to four neighbors as default. Table I and II summarize the traces we have collected. We are installing this client in several places around the world and gathering traces for extended durations.

Location	Trace length	Number of IPs
CMU	10 hours	799,386
ATT	14 hours	302,262
ACIRI	6 hours	185,905

TABLE I
GNUTELLA TRACES WITH UNLIMITED CONNECTIONS.

Location	Trace length	Number of IPs
CMU	89 hours	301,025
ATT	139 hours	261,094
WPI	10 hours	69,285
UKY	96 hours	409,084
UKY	75 hours	292,759

TABLE II
GNUTELLA TRACES WITH LIMITED CONNECTIONS.

We observed that some clients had used private IP addresses in the collected Gnutella traces. These private IP addresses are in the following ranges: 10.x.x.x, 172.16.0.0 - 172.31.255.255, and 192.168.x.x. Since private IP addresses are designed for use on internal networks and cannot be clustered using network-aware clustering, we remove them from the traces before applying network-aware clustering and present our results below. There are 8% - 16% of all IP addresses identified as private IPs in the traces.

We clustered the user IP addresses extracted from the traces. Figure 1 plots the distributions of the number of hosts and the number of messages of the CMU trace (we show results from one trace, other results are similar). The distribution of the number of hosts in a cluster is non-uniform: more than half of the clusters have a small number of clients, with some issuing a large number of messages. Figure 2 plots the client and cluster distributions observed every 30 minutes. The number of clients observed in each 30-minute period varies between 5% - 10% of all clients in the trace, which implies that the Gnutella network is very dynamic, with peers joining and leaving frequently. The percentages of clients observed in the trace during each 30-minute period is much smaller than that of clusters, indicating the number of clusters in the network is more stable. Thus, network-aware clustering based scheme helps reduce dynamism in the P2P network.

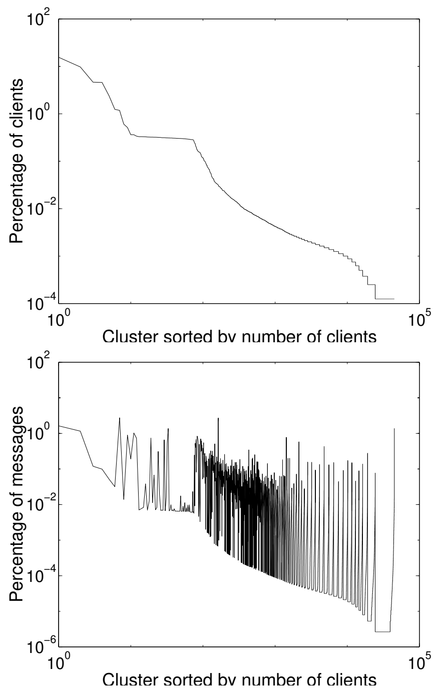

Fig. 1. The cluster distribution of the CMU trace (in log scale). The clusters are sorted by number of clients.

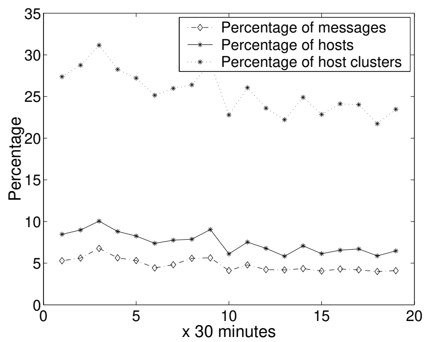

Fig. 2. The client and cluster distribution per 30 minutes in the CMU trace.

The distribution of replies observed in the trace is generally proportional to the number of clients within a cluster. We noticed several exceptions where small clusters generate a large number of replies. Two possibile reasons for this are: (1) The cluster has a large number of files and is able to reply to many of the search requests. (2) The cluster has a few popular files that are requested by many clients. Because a *ping reply* Gnutella message tells us how many files a host is willing to share, we extract the IP addresses from *ping reply* messages and match them to those in the *search reply* messages. There are 54,743 such IP addresses matched in the CMU trace. We observe that clusters with more files usually generate more replies. There are also some clusters with smaller number of files that generate a

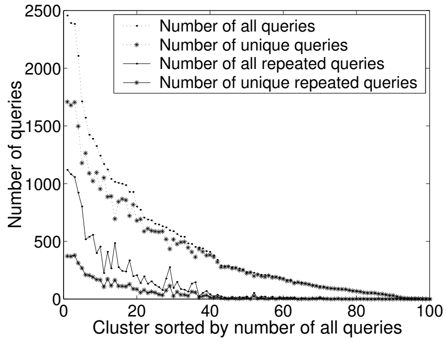

Fig. 3. Query distribution using network-aware clustering

lot of replies, implying the existence of popular files.

We observe that there are popular queries in the Gnutella traces which can be cached to reduce query latency. We sample the CMU trace and take the first 47,100 *search request* messages with 8,878 unique queries. Up to 26% of the unique queries from each cluster are submitted more than once; while up to 48% of all queries from each cluster are repeated ones and can be cached. Figure 3 plots the number of queries and repeated queries from each cluster. A query is considered to be from a cluster if a user within the cluster either submits or forwards the query (we cannot differentiate query initiator from the client who forwarded the query based on the Gnutella message). A query is considered to be repeated if it is observed more than once in the trace.

We compare the performance of CAP and Gnutella via a trace-driven simulation. We assume the user joins the network when its IP address appears in the trace for the first time. If a user does not send a message for a certain period of time, we assume the user has left. We extract filenames from the *search reply* messages and assign them to the corresponding users. We generate queries by randomly picking users requesting for files existing in the system. We also assume the query popularity distribution follows the file popularity distribution, i.e., a file is queried multiple times if it appears in the multiple replies in the Gnutella traces. This assumption is validated by examining the query popularity distribution and the file popularity distribution using the same trace. We sample the first 10,000 nodes in the CMU trace and Figure 4 plots the two distributions.

To forward queries, each delegate node keeps a list of neighbor delegate nodes and their cluster prefixes. Each query also has a maximum search depth. If a query cannot be resolved at the delegate node, it will be forwarded to the neighbors in the list using the depth-first search algorithm until the object is found or the delegate node has tried all

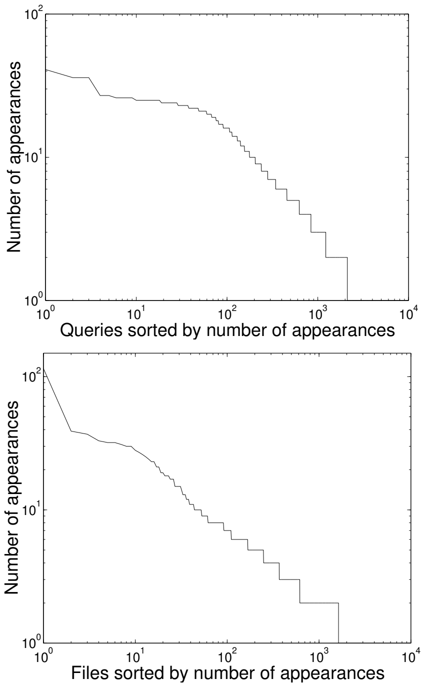

Fig. 4. Query popularity distribution and file popularity distribution

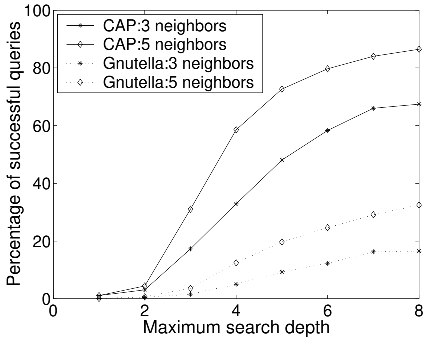

Fig. 5. Percentage of successful queries using the CAP algorithm and the Gnutella algorithm.

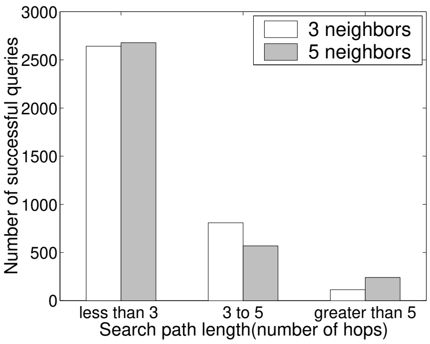

Fig. 6. Search path length of successful queries in CAP.

its neighbors in the list. The neighbor list can be obtained initially from the clustering server and be improved gradually based on the application requirements. In our simulation, the neighbor list is assigned randomly when the delegate node joins; it is then updated with the neighbor who finds the largest percentage of requested objects ranking first. This is a heuristic based on the assumption that the neighbor who finds the most number of requested objects is likely to find more objects in the future. Other ways to improve the neighbor list could involve use of search latency. For simplicity, we assume current directory information is stored at each delegate node.

Since CAP does not guarantee that an existing object will be found, we examine the probability for finding an object. Again, using the CMU trace to illustrate the results, we consider a network of 1,000 nodes with 311 clusters. There are 4,615 queries in our trace requesting 3,793 unique files. Each data point in our graphs is the average of 10 runs. Figure 5 plots the percentage of successful queries by varying the maximum search depths in CAP and in Gnutella. We observe that the percentages of successful queries in CAP are much higher than that in Gnutella. We measure the search latency in terms of the number of hops traversed before an object is found with a fixed maximum search depth of five hops. Figure 6 shows the search path length of successful queries using the CAP algorithm. Most of the successful queries are resolved within twenty hops, indicating that a query message will not affect many nodes in the network, as opposed to Gnutella, which will affect all the nodes in the tree rooted at the initiating node.

Figure 7 plots the average number of messages due to each query and the average number of forwarding operations performed at each delegate node in CAP or node in Gnutella algorithm, respectively. Because queries are flooded in Gnutella, the number of messages per search and the number of forwarding operations performed per node grow exponentially with search depth; while in CAP, both of them grow linearly.

IV. Conclusion and future work

We have analyzed Gnutella traces using network-aware clustering. We outlined the design of a Cluster-based Architecture for P2P systems and our early measurements show that CAP is scalable and stable. We are carrying out a broad study of P2P traces using network-aware clustering and plan to examine the performance of CAP based on real deployment. We are also examining a number of issues such as routing queries within and between clusters, handling updates, as well as how cluster diameters, latencies, and workload affect the performance and P2P traffic.

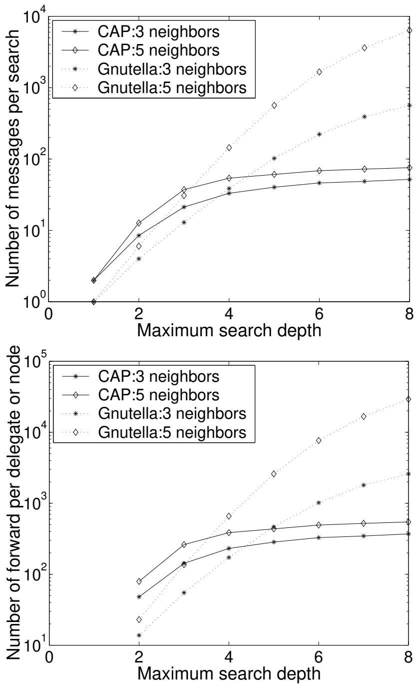

Fig. 7. Number of messages per search and number of forwarding operations performed by each delegate in CAP or node in Gnutella.

References

[1] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies:International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2000*, December 2000.

[2] Gnutella hosts. http://www.gnutellahosts.com.

[3] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *Proceedings of ACM Sigcomm*, August 2000.

[4] Napster. http://www.napster.com.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM Sigcomm*, August 2001.

[6] A. Rowstron and Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *under conference submission*, February 2001.

[7] GNUT-gnutella. `http://www.gnutelliums.com/linux_unix/gnut/`.

[8] K. Sripanidkulchai. The popularity of Gnutella queries and its implications on scalability. In *The O'Reilly Peer-to-Peer and Web Services Conference*, September 2001.

[9] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM Sigcomm*, August 2001.

[10] Y. Zhao, John D. Kubiatowicz, and Anthony Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2000.