CHORELLA – A Reliable and Scalable Addressing Scheme for Data Distribution

Sravanthi Peruru

***Abstract*— The core problem confronting Distributed Data Storage and peer to peer applications is to efficiently locate the node where the requested data is present. This paper presents Chorella, an addressing scheme to solve this problem. Chorella, which is built upon the ideas of Chord and Gnutella (peer to peer protocols) finds the location of the requested data in the network. Chorella can adapt efficiently when the existing nodes leave the network under certain conditions.**

***Index Terms*—Addressing Scheme for data distribution, Chorella.**

I. Introduction

Data centers store, manage, process, and exchange digital data and information. In the general case, the data may be distributed over many servers. Peer to peer systems are loosely organized systems without any centralized control or any hierarchical structure. Data centers and peer to peer systems share a common communication model in which each node acts as both client and server. A node's request is forwarded to all or few other nodes in the network until the desired node which contains the requested data is found.

II. Problem Formulation

The currently popular peer to peer protocols, such as Chord and Gnutella, suffer from serious drawbacks. In case of Chord, data needs to be moved among the nodes continiously as nodes join and leave the networks , implying it to be unacceptable. And with Gnutella, the request traffic is too high. In view of these drawbacks, we propose a new protocol called Chorella which provides a reliable and secured addressing scheme.

The Chord protocol will resolve all the lookups via O(log N) messages to other nodes[1]. There can be a more efficient lookup service which can take less than O(log N) messages. Also the stabilization overhead in Chord prevents it from being used widely. Although the Gnutella protocol is a widely used model, has many drawbacks. The main drawback of Gnutella protocol is its heavy request traffic. The passing of messages in Gnutella generates much traffic in the network often leading to network congestion and slow responses.

SravanthiPeruru is with the Computer Science Department, Oklahoma State University, Stillwater, OK 74078 USA. (e-mail: sravanthi.peruru@okstate.edu)

And this makes the quality of service poor, since the response to queries are delayed. A study [3] showed that the traffic in Gnutella systems is mainly due to messages for establishing intial connections and queries. Ripeanu [4] reported that the traffic generated in Gnutella consists of approximately 92% Query messages, 8% Ping messages and hence the other messages constitute less than 1% of the traffic.

III. Background

Chord and Gnutella are the most prominent peer to peer protocols. The proposal of Chorella has been motivated by these two protocols Chord and Gnutella. Chord is a peer to peer protocol which presents a way out, to the problem of efficient location of a node. Chord supports just one operation: given a key, it maps the key onto a node [1]. Chord uses the routed queries to locate a key with a small number of hops, which remain considerately small even if the system contains a large number of nodes. Chord uses consistent hashing to assign keys to the nodes. Each node and key is assigned an m-bit identifier using SHA-1 as a base function. Node ID (identifier) is obtained by hashing the IP address of the node. KeyID (identifier) is obtained by hashing the filename. Length of the identifier should be large enough so that the probability of two node IDs or keyIDs hashing to the same identifier is negligible. In consistent hashing keys are assigned in the following way: Identifiers are ordered on an identifier circle modulo 2^m. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. The node is called the successor node of key k, denoted by successor(k).

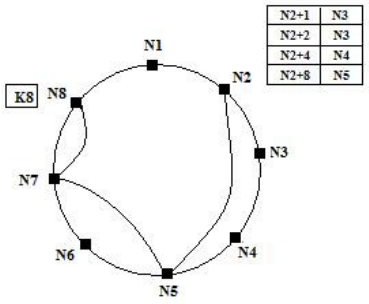

Fig. 1. Routing through Chord circle.

The principal function of Chord protocol is the key look-up function. Every node n maintains a routing table with up to m entries (where m is the number of bits of the identifiers) called

finger table. The i^{th} entry in the table at node n contains the identity of the first node S that succeeds n by at least 2^i-1 on the identifier circle. This node S is called the ith finger of node n. When a node is asked to find a file associated with a particular key, it will first find the highest successor of this key in its routing table and forward the key to that node. This process continues until the node responsible for that key is found. Because each node has finger entries at power of two intervals around the identifier circle, each node can forward a query/request at least half way along the remaining distance between the node and the target identifier.

Fig.1. Shows how the routing takes place in traditional Chord.

Gnutella is another common protocol for filesharing P2P applications [2]. A node that is posed with a request sends the query to all of its neighbour nodes, who will inturn send the query to all of their neighbour nodes and this query broadcasting continues until the query reaches a node that has a file which matches the query, or until a certain predefined maximal number of forwards is reached. If the destination node (node where the requested file is present) is reached, then that node sends back a reply to the query containing its address, the size of the file, speed of transfer, etc. The reply travels through the same path by which the query arrived, but in the reverse order back to the node that posed the query. In this manner each message is propogated to upto n^p other nodes where n is the number of neighbour nodes and *p* is the maximum number of forwards called "time to live" (TTL) of the query.

Fig. 2 represents the communication that takes place in Gnutella. When node 1 recieves a query, it broadcasts the query to its neighbour nodes 2 and 3. 2 and 3 nodes in turn broaddcast the request to their respective neighbours which are 4, 5 and 6, 7 respectively. Nodes 4, 5, 6, 7 in turn broadcast their requests to their neighbours. In this way the request traverses the network until it hits a nodes with the required response or until a maximal number of forwards is reached.

IV. Proposed Approach

The prime goal behind designing the new protocol CHORELLA is to provide an efficient way of addressing a node in the distributed data storage systems and to avoid the heavy network traffic and minimize the number of lookup messages. CHORELLA steals few of the properties of Chord. It is similar to Chord in a way how the Chord organizes its nodes on an identifier space. DNS provides a host name to IP address mapping [8]. CHORELLA can provide the same service with the name representing the key and the associated IP address representing the value. We take into consideration that there can be a maximum of 2^{20} nodes that can be a part of the network; Chorella identifies a 2^{20} bit identifier space to place the nodes on the identifier circle.

A. Mapping of nodes

Every node that wants to be a part of network, must register itself with the manager node. The manager node assigns it a 20 bit ID. The manager node also provides each node that registers, with a copy of mapping table.

Let this 20 bit address assigned by the manager node be the nodeID of the corresponding node.

Place this node on an identifier circle of identifier space 2^{20}. Identifier circle can be thought of as a circle which represents values between 0 and 2^{20}-1

Each node stores a copy of mapping table. Mapping table contains a set of IP addresses and their corresponding 20 bit mapped value, of all the nodes present in the network.

Fig. 3 shows how a node is placed on an identifier circle. When a node with nodeID 6 wants to join the network, it is placed at an identifier space 6 on the identifier circle.

Table I illustrates the mapping table which holds the IP addresses and their corresponding nodeIDs (represented in 6 bits).

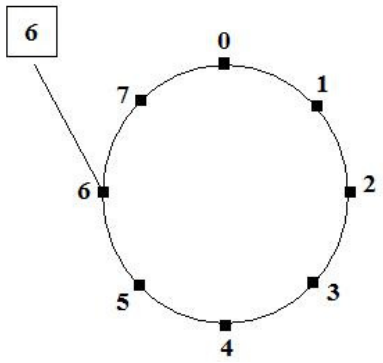

Fig. 3. Node placement on an identifier circle with identifier space 2^3.

B. File Distribution in CHORELLA

Any file that is to be placed on a node:

Hashes it keys (Key of the file depends upon the application using it, Chorella doesn't define this) to a 160 bit hash value using SHA-1[7], a consistent hashing [5], [9] algorithm.

Least significant 20 bits of the 160 bit hash value, together with 160 bit hash value and filename forms the KeyID.

KeyID = [least significant 20 bits of the 160 bit hash value ; 160 bit hash value of key ; filename]

Though the 20 bit key hash of each file may not be unique but, the 160 bit hash value together with 20 bits and file name makes it unique.

This file is placed at a node whose nodeID is equal to or follows the least significant 20 bits of the 160 hash of the filename.

C. File Search in CHORELLA

When a client (one among 2^{20} nodes) requests for a file:

The client itself converts the key associated with the file into a 160 bit hash value using SHA-1 algorithm.

It extracts the least significant 20 bits of the 160 bit hash value and forms the keyID

TABLE I
MAPPING TABLE

IP Address	6 bit mapped value
139.78.67.161	000000
19.178.197.01	000001
248.216.90.87	000010
32.7.165.987	000011
19.148.27.1	000100
27.68.69.178	000101
169.218.197.1	000110
247.158.67.87	000111

Mapping table with IP address and its corresponding 6 bit mapped value (nodeID).

The client then obtains the IP address of the node whose nodeID is equal to the first value (20 bits) of the keyID searching from the mapping table. It searches for the IP address corresponding to the node whose nodeID is equal to the 20 bits in keyID.The client then requests that node for that particular file.

Fig. 4. Represents an example for the file search operation. When there is a query for file name db.txt (i.e., a file, named db.txt is requested by the client), the client will first convert the key of the file into an integer value of length 4 (In case of CHORELLA it is the 160 bit hash value) which is 6202. The least significant 2 digits of the 6202, i.e., 02 is extracted. The client now knows that the file should be present with the node whose nodeID is equal to 02. So the client now searches for the IP address corresponding to nodeID 02 from the mapping table and obtains it. After obtaining the IP address the client

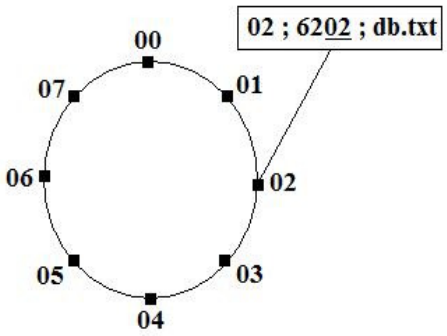

Fig. 4. File search in CHORELLA

directly sends a request to the node whose IP address is obtained, for a file named db.txt.

D. Node Failure in CHORELLA

To ensure the data availability even in case of node failure:

Each file on a node is replicated [6] and placed at other nodes.

File is stored at a node whose nodeID is equal to or follows the keyID in the identifier circle. Every file is replicated and stored at two or more locations.

Copy of the file is stored at locations or nodeID's which are obtained by rehashing the 160 bit hash value of the original 160 bit hash value and extracting the least significant 20 bits from it. This mirror file is stored at a node whose nodeID is equal to these 20 bits, obtained. No of replications or the no of mirrors depend on the probability of node failure in the network. The client first requests the owner node for the file.

If the owner doesn't respond to the request submitted by the client in a stipulated time interval, the client forwards its request to that node whose nodeID is obtained by rehashing the 160 hash of the file key. In this manner the client searches for the file with every mirror until it finds the requested file. This makes the protocol reliable even in case of node failure.

There is a manager node which is always alive, to keep track of modifications in mapping table. If any client submits its request and the node doesn't respond within some stipulated time interval, the client assumes that node to be absent in the network and thus reports to the manager node about the absence of that node. The manager node will update its mapping table by removing the tuple corresponding to the node that is left.

The manager node, circulates its mapping table to all the nodes in the network after some time interval, in order to stabilize the network by updating the mapping tables present with every node in the network and thus makes the network stable.

V. CONCLUSION

Many applications need to determine a node that stores a data item. Chorella performs an action of finding a node given a file in a single hop. Striking features of Chorella includes scalability, reliability. Such a system could also be used for storing pieces of data on several nodes.

.

REFERENCES

[1] MORRIS, R., KAASHOEK, M. F., KARGER, D., BALAKRISHNAN, H., STOICA, I.,LIBEN-NOWELL, D., DABEK, F.: Chord: A scalable peer-to-peer lookup protocol for internet applications http://pdos.csail.mit.edu/papers/Chord:sigcomm01/Chord_sigcomm.pdf

[2] GNUTELLA Website: http://gnutella.wego.com.

[3] Vaucher, J., Babin, G., Kropf, P., and Jouve, Th. (2002), "Experimenting with Gnutella Communities", Distributed Communities on the web (DCW 2002), Sydney, Australia. LNCS 2468, Springer Berlin, pp.85-99

[4] Ripeanu, M. (2001), "Peer-to-Peer Architecture Case Study: Gnutella Network", Proceedings of IEEE 1st International Conference on Peer-to-Peer Computing Linkoping , Sweden.

[5] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN,D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, TX, May 1997), pp. 654–663.

[6] PLAXTON, C., RAJARAMAN, R., AND RICHA, A. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM SPAA* (Newport, Rhode Island, June 1997),pp. 311–320.

[7] FIPS 180-1. *Secure Hash Standard.* U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[8] MOCKAPETRIS, P., AND DUNLAP, K. J. Development of the Domain Name System. In *Proc. ACM SIGCOMM* (Stanford, CA, 1988), pp. 123–133.

[9] LEWIN, D. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of EECS, MIT, 1998. Available at the MIT Library, http://thesis.mit.edu/.