Volume 3, Issue 1, January 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Protocol to Find Multimedia Data in Peer-To-Peer Network

Mohanjeet Singh*
Research Scholar
PTU Jalandhar, Punjab,
India

D. S. Dhaliwal
Bharat Group of Colleges
Sardulgarh, Punjab,
India

Neeraj Garg
Haryana Engineering College
Jagadhri, Haryana,
India

Abstract: Peer-to-Peeris an Application-layer protocol.To form and maintain an overlay among participant nodes peer to peer protocol can be used.Search for a resource-object in the overlay it provides mechanisms for nodes to join, leave, publish. How does a node know which node in the system contains the data it is searching for is the core problem of peer-to-peer applications? That means efficient location of nodes is the main problem. To address this problem several solutions have been found. A protocol with higher standards is needed in order to use peer-to-peer systems for more sophisticated applications such as business applications. In this paper we have discuss some protocols which provide the solution to find the node containing the required data.

Keywords--Peer, Chord, Can, Tapestry, Link State

I. INTRODUCTION

Routing tables are used by Peer to Peer protocolsto maintain information about nodes.Node's routing table only contains a subset of these nodes because nodes contain large content. If a node can't accomplish the request, it searches for one who can, by performing a nextHop () operation to a destination from the routing table (recursive-routing).A common aspect of these operations is to find an appropriate node in the overlay. Routing table which contains the description of another node is used by node to find an appropriate node.There are three high-level requirements for a peer-to-peer protocol. Resource publishing and lookup: The protocol should provide a mechanism for a peer to publish a resource-object or advertise its service and a mechanism to lookup the resource-object and the node offering a service. P2P network maintenance: In a peer-to-peer network the protocol should provide mechanisms to maintain connectivity and resource availability. Heterogeneous connectivity:In heterogeneous network environments Nodes should be able to form an overlay and exchange information about their uptime and capacity.

II. RELATED WORK

The TRIAD [9] project at Stanford University focuses on the problem content distribution, integrating naming, routing and connection setup into its content layer. TheGlobe [10] project was one of the first location mechanisms to focus on wide-area operation. It used asmall fixed number of hierarchies to scale location data, making it unable to scale to increasingly large networks. The Distance Vector (DV) and Link State (LS) algorithms used in IP routing require every router to have some level of knowledge of the topology of entire network. DV and LS thus require the widespread disseminationof local topology information. While well suited to IP networkswherein topology changes are infrequent, for networks with frequenttopology changes, DV and LS would result in the frequentpropagation of routing updates.

III. CHORD

Chord is a peer-to-peer protocol.It is used to presents a new approach to the problem of efficient location. Chord from other applications is its simplicity, its provable performance and provable correctness. Chord gives a key and maps the key onto a node. Data localization can be implemented by associating each key with a data item.Chord software interacts with the application which is in C++ in the form of library consists of 3000 lines. The software interacts with the application in following ways:

• It provides a lookup (key) – function, which yields the IP address of the node responsible for the key.
• It notifies the node of changes in the set of keys the node is responsible for.

A. Assets Of Chord

1) *Decentralization:*Each node acts as a server or clientand thereis no central server or super peer.This property provides robustness and it does not have a single point of failure.
2) *Availability:* The protocol work well even if the system is in a continuous state of change: Despite major failures of the underlying network and despite the joining of large number of nodes, the node responsible for a key can always be found.
3) *Scalability:* The cost of a Chord lookup grows only logarithmically in the number of nodes in the system, so Chord can be used for very large systems.
4) *Load balance:*Chord uses a consistent hash function to assign keys to nodes. Therefore, the keys are spread evenly over the nodes.

5) *Flexible naming:*Chord imposes no constraints on the key structure, so the user is granted a large amount of flexibility in the data can be named.

B. Chord Protocol

The protocol contains functions to locate nodes and to deal with joins and failures of nodes.

1) *The Chord Ring:* The Chord protocol uses SHA-1 [2] as consistent hash function to assign m-bit identifier to each node and each key. Consistent hash functions are hash functions with some additional advantageous properties, i.e. they let nodes join and leave the system with minimal disruption [3][4]. The m is an integer which should be chosen big enough to make the probability that two nodes or two keys receive the same identifier negligible.

The hash function calculates the key identifier by hashing the key, and the node identifier by hashing the IP address of the node. The key and the node identifiers are arranged on an identifier circle of size 2^m called the Chord ring. The identifiers on the Chord ring are numbered from 0 to 2^{m-1}. A key is assigned to a node whose identifier is equal to or greater than the identifier of the key. This node is called the successor node of k, denoted by successor (k), and is the first node clockwise from k on the circle.

2) *Key Location:*The core function of the Chord protocol is the key location function. For a better understanding, a simple key location function is introduced first. Next, the scalable key location function will be demonstrated.

3) *Simple Key Location*: In order to let a node n find a certain key k, we call n.lookup(k). To execute the lookup, the protocol will call the function find_successor, which will return the successor node from node n if k lies between n and its successor or forward the query around the circle otherwise. In the worst case, the query needs to be forwarded N times in a circle with N nodes, so the cost of a lookup is linear in the number of nodes.In systems with a large number of nodes, lookups would be too slow. Therefore, Chord uses a scalable key location function which will provide more efficient lookups.

4) Scalable Key Location: In order to provide more efficient lookups, additional routing information is stored to accelerate lookups. Each node n maintains a routing table with up to m entries (where m is the number of bits of the identifiers) which is called the finger table.The ith entry in the table at node n contains the first node s that succeeds n by at least 2i-1 This node s is called the ith finger of node n.

   Important characteristics of this scheme are:

   Each node stores information about only a small number of nodes (m).

   Each node knows more about nodes closely following it than about nodes farer away.

   A finger table generally does not contain enough information to directly determine the successor of an arbitrary key k. A node has to contact other nodes in order to resolve the hash table.

   When a node is asked to find a certain key, it will determine the highest predecessor of this key in its routing table and forward the key to that node. This procedure will recursivley determine the node responsible for the key. The lookup time is O(log N), since the query is forwarded at least half the remaining distance around the circle in each step .

5) *Joining of Nodes:*When a node joins the system, the successor pointers of some nodes will have to change. It is important that the successor pointers are up to date at any time because the correctness of lookups is not guaranteed otherwise. The Chord protocol uses a stabilization protocol running periodically in the background to update the successor pointers and the entries in the finger table.

   When nodes have joint recently and a lookup occurs before stabilization has finished, the system finds itself in one of these three states:

   All finger table entries and successor pointers are correct.

   The successor pointers are correct, but the finger table entries are not.

   The lookup will still be correct, but might be a little slower

   In case a larger number of nodes have joined between the target and the target's predecessor, the find_successor function will initially undershoot and some of the hops will be in linear time.

   Neither finger table entries nor successor pointers are correct:In this case, the lookup will fail. The higher-layer software using Chord will notice that the data was not found and will retry after a short pause.

6) *Impact of node joins on performance:*When stabilization has been completed, there is no impact onperformance beyond increasing N (total number of nodes) in the O (log N) lookup time.

   When stabilization has not been completed, the lookup speed might be affected if nodes join between the target and the target's successor. But this is only the case if the number of joining nodes is very large. In general, it can be stated that lookups take O(log N) hops as long as the time it takes to adjust finger tables is less than the time it takes the network to double in size.

7) *Failure of Nodes:*The correctness of the Chord protocol relies on the fact that each node knows its successor. When nodes fail, it is possible that a node does not know its new successor, and that it has no chance to learn about it. To avoid this situation, each node maintains a successor list of size r, containing the node's first r successors. When the successor node does not respond, the node simply contacts the next node on its successor list.

Assuming that each node fails with a probability p, the probability that every node on the successor list fails is p^r. Increasing r makes the system more robust. With this parameter tuning, any degree of robustness can be achieved.

It can be proven that under the assumption that the network is initially stable, and every node fails with probability ½, find_successor still finds the closest living successor to the query key and the expected time to execute find_succesor is O(log N) [1]. Simulation results have shown that even massive failures have little impact on robustness.

IV. CONTENT-ADDRESSABLE NETWORK (CAN)

The peer-to-peer file transfer process is inherently scalable, but the hard part is finding the peer from whom to retrieve the file. Thus, a scalable peer-to-peer system requires, at the very least, a scalable indexing mechanism. We call such indexing systems Content-Addressable Networks. Before CAN most of peer-to-peer designs are not scalable. Central server stores the index of all the files available. To retrieve a file, a user queries this central server using the desired file's well-known name and obtains the IP address of a user machine storing the requested file. The file is then down-loaded directly from this user machine. This makes it both expensive (to scale the central directory) and vulnerable (since there is a single point of failure). Content-Addressable Network (CAN) provides hash table-like functionality. The CAN is scalable, fault-tolerant and completely self-organizing. Ahash table is a data structure that efficiently maps "keys" onto "values" and serves as a core building block in the implementationof software systems.

A. Designing and Routing in a CAN

Nodes in the CAN self-organize into an overlay network that represents this virtual coordinate space. A node learns and maintains the IP addresses of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbors in the coordinate space serves as a coordinate routing table that enables routing between arbitrary points in this space. Routing in a Content Addressable Network works by following the straight line path through the Cartesian space from source to destination coordinates. A CAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zone of each of its immediate neighbors in the coordinate space. In a d-dimensional coordinate space, two nodes are neighbors if their coordinate spans overlap along d-1 dimensions and abut along one dimension. This purely local neighbor state is sufficient to route between two arbitrary points in the space: A CAN message includes the destination coordinates. Using its neighbor coordinate set, a node routes a message towards its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. Note that many different paths exist between two points in the space and so, even if one or more of a node's neighbors were to crash, a node can automatically route along the next best available path.

B. *Bootstrap and Zone Finding*

A new CAN node first discovers the IP address of any node currently in the system. As in [5] we assume that a CAN has an associated DNS domain name, and that this resolves to the IP address of one or more CAN bootstrap nodes. A bootstrap node maintains a partial list of CAN nodes it believes are currently in the system. Simple techniques to keep this list reasonably current are described in [5]. To join a CAN, a new node looks up the CAN domain name inDNS to retrieve a bootstrap node's IP address. The bootstrap nodethen supplies the IP addresses of several randomly chosen nodescurrently in the system.

The new node then randomly chooses a point P in the space and sends a JOIN request destined for point P. This message is sent into the CAN via any existing CAN node. Each CAN node then uses the CAN routing mechanism to forward the message, until it reaches the node in whose zone P lays.

V. TAPESTRY

It is a self-organizing, scalable, robust wide-area infrastructure that efficiently routes requests to content, in the presence of heavy load and network and node faults. Providing location-independent routing of messages directly to the closest copy of an object or service using only point-to-point links and without centralized services. Tapestry employs randomness to achieve both load distribution *and* routing locality. It has its roots in the Plaxton distributed search technique [6], augmented with additional mechanisms toprovide availability, scalability, and adaptation in the presence of failures and attacks. Tapestry is selfadministrating,fault-tolerant, and resilient under load, and is a fundamental component of the OceanStoresystem [7, 8].

A. Fault-Tolerance, Repair, and Self-Organization

Topology of the location and routing infrastructure must be self-organizing, as routers, nodes, anddata repositories will come and go, and network latencies will vary as individual links fail or vary their rates. Thus, operating in a state of continuous change, the routing and location infrastructure must be able to adapt the topology of its search by incorporating or removing routers, redistributing directory information, and adapting to changes in network latency. The large numbers of components suggeststhat any adaptation must be automatic, since no reasonable-sized group of humans could continuously tunesuch an infrastructure.

The intuition of the incremental algorithm is as follows: First, we populate the new node's neighbor mapsat each level by routing to the new node ID, and copying and optimizing neighbor maps along each hopfrom the router. Then we inform the relevant nodes of its entry into the Tapestry, so that they may update their neighbor maps with it.

Another Tapestry goal is to provide an architecture that quickly detects environmental changes and modifies node organization to adapt. First, changes in network distance and connectivity between node pairs drastically affect overall system performance. Tapestry nodes tune their neighbor pointers by running a refresher thread which uses network Pings to update network latency to each neighbor.

VI. CONCLUSIONS

Protocols discuss above are power full protocols which solve the problem of efficient data location. Chord scales well with number of nodes what makes it an interesting application for larger systems. Chord continues to function correctly even if the system undergoes major changes and if the routing information is only partially correct. The CAN is completely self-organizing, fault-tolerant and scalable. Some problems remain to be addressed in realizing

a comprehensive CAN system. An important open problem is thatof designing a secure CAN that is resistant to denial of service attacks.This is a particularly hard problem because a malicious node can act, not only as a malicious client, but also as a malicious server or router. Tapestry is self-organizing, scalable, robustwide-area infrastructure that efficiently routes requests to

content in the presence of heavy load andnetwork and node faults. Tapestry provides an ideal solution to deliver messages usingonly point-to-point links and without centralized services.

Acknowledgment

I would like to thanks Punjab Technical University Jalandhar, for providing me opportunity and technical support to complete this research work.

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,*To Appear in IEEE/ACM Transactions on Networking. http://www.pdos.lcs.mit.edu/chord/

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

[3] Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., And PanigrahyR,*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.

[4] 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/.

[5] P. Francis. Yoid,*Extending the Internet Multicast Architecture,* Unpublished paper, available athttp://www.aciri.org/yoid/docs/index.html, Apr. 2000.

[6] C. Greg Plaxton, RajmohanRajaraman, and Andrea W. Richa,*Accessing nearby copies of replicatedobjects in a distributed environment.* In Proceedings of ACM SPAA. ACM, June 1997.

[7] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi,Sean Rhea, HakimWeatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao,*OceanStore: Anarchitecture for global-scale persistent storage*,In Proceedings of ACM ASPLOS. ACM, November2000.

[8] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon, and John Kubiatowicz.Maintenance,*free global storage in OceanStore,* Submission to IEEE Internet Computing,2001.

[9] Mark Gritter and David R. Cheriton,*An architecture for content routing support in the internet*,InProceedings of the Usenix Symposium on Internet Technologies and Systems. Usenix, March 2001.

[10] Maarten van Steen, Franz J. Hauck, Philip Homburg, and Andrew S. Tanenbaum,*Locating objects inwide-area systems*. IEEE Communications Magazine, pages 104–109, January 1998.

