CS 856: Paper Summaries
W Anthony Young - 20161423
October 14th, 2004

Stoica, et. al - Chord

Summary of Contents

This paper begins with an introduction of the Chord protocol and how it is novel. Chord supports only the function of mapping keys to values using a fixed hash function. Chord maintains only O(log N) routing data, and any message can be sent to any node in only O(log N) hops.

After presenting a brief introduction of the Chord system, the paper turns to relate Chord to other works. For example, Chord is similar to DNS in that it maps names to values. It is also similar to Freenet in that it is decentralized and adapts easily to new peers joining the network. As well, since Chord does not broadcast messages as Gnutella does, it experiences much better scalability.

The paper next turns its attention to the system model of Chord. The Chord protocol is implemented as a library function accessed by client systems. The library supports lookup of values from keys. Chord provides load balancing by spreading keys over nodes, decentralization of lookup, scalability of retrieval, availability of data when nodes are online, and flexible naming of keys. As such, Chord is ideally suited to many applications including cooperative mirroring of software, time shared storage of data, distributed indexing of available files for file sharing systems, and combinatorial searches, to name a few.

Next, the authors turn their attention to a discussion of the Chord protocol. Chord makes use of consistent hashing, where each node is assigned an identifier by hashing its IP address. Nodes are then inserted into an identifier circle modulo 2^m where m is the number of bits in the chosen key value. When values are hashed, they are assigned to the node in the circle with identifier equal to, or greater than, the hashed value. In order to lookup a node, a client consults its finger table to find the first node after the key they are looking for, modulo 2^m (again). Thus, only log N table entries are needed to hold all the lookup information. Hence, Chord scales very well.

The authors then turn their attention to the algorithm for nodes joining the network. To join the network, a node must find its place in the circle, initialize its finger table and predecessor link, update the finger table entries for each other node in the system, and transfer all the data it is now responsible for to it. The new node can find its predecessor by asking a known node to look it up. Then, the new node can use its predecessor's finger table to construct its own. The node must update existing finger tables by walking around the circle, and finally, transfer all the values for the keys that it is now responsible for by invoking some application process that downloads the data or values associated with its new keys, and removes them from the current holders.

The article then turns to discuss how several nodes can join the network at once, and how finger tables can be stabilized in the face of random failures. When a node notices that its successor has failed, it can replace its link to a successor with another selection from a successor list, thus maintaining the ring structure. The node then invokes the stabilization procedure to keep its finger table up to date and to remove the failed node.

As the article draws to a close, the authors highlight some performance analysis. As presented earlier, the authors verify that only O(log N) hops are needed to perform a query lookup, and only O($\log^2$ N) entries are needed in a finger table to reach all N nodes in a network. The authors performed their analysis using simulation systems instead of actual implementations. The authors conclude the analysis by presenting evidence from prototype implementations that Chord's lookup latency grows slowly with the total number of nodes, thus confirming the simulation results of O(log N) hops.

Finally, the authors present some future directions to take their work, and conclude with a short summary.

Novelty

This paper presents a scalable lookup protocol for key/value pairs in a peer-to-peer system. The protocol is novel as it scales less than linearly with the number of nodes in the system, both in routing cost and routing table size.

Organization

The information in this paper is well organized into relevant sections. However, it would be useful to present the entire data structure and system architecture for Chord in the system architecture section. The authors provide a brief overview of the system, but leave several details to be discussed and presented in future sections. For example, discussion of the successor list is left until section 5. It would be useful to me to have a complete picture of the system described before details are given.

Writing Style

This paper is generally well written, but some sections are wordy and very technical in nature. I would have appreciated more explanations and elaboration on some sections, particularly those involving lookup and finger table stabilization.

Content

In general, the authors do a good job describing all the relevant information in this paper. In fact, the only problem I can see with this protocol is the necessity to move data from node to node when a new node takes over some portion of a key space. If, for example, Chord is being used to hold GIS data, it is possible that large amounts of data will need to be transferred when a node joins or plans to leave a network. This could take a significant amount of time to accomplish and slow the network quite a bit.

Ratnasamy - A Scalable Content-Addressable Network

Summary of Contents

This paper begins with a background to peer networks. The author discusses the history of peer networks and why they are popular today. After presenting some of the challenges that peer systems currently face (i.e. file lookups, etc.), the author introduces the idea behind CAN: a routing protocol called the Content Addressable Network. CAN runs on top of a DHT-based peer network in order to provide its users with fast file lookups and reduced overhead traffic.

Once the background discussion is complete, the author presents the CAN protocol in depth. CAN nodes organize themselves into a d-dimensional space automatically. Messages are then deterministically routed through the overlay to a "zone" (key range). Each node maintains a list of its neighbours and the zones that they are responsible for. In order to join the network, a node finds a peer though a bootstrap process, splits the largest zone in the system, takes control of the split key space, and updates the DHT's of its new neighbours. Nodes maintain an ID (VID) that details their key space, and must only maintain 2d VID's of neighbour peers. This makes CAN scalable in space used. Since the network is organized logically into a torus, routing a message takes $(d/4)(n^{1/d})$ hops. Messages are forwarded with a greedy algorithm to the neighbour with the closest key range to the key being searched for. When a node leaves the system, its space is taken over by the node with the numerically closest VID. Algorithms for joining or leaving a network, splitting a zone, and failure recovery are presented.

The author then spends time analyzing the CAN protocol. CAN scales very well, as for n nodes, the maximum path length in neighbours is $O(dn^{1/d})$. As well, increasing the dimension above log(n/2) doesn't reduce the path length as n nodes are insufficient to fill the dimension space. CAN is very resilient to random failures as there are many redundant paths created through the partitioning scheme. As the dimension of the network increases, so does the resilience, as there are even more redundant paths through higher dimension hops. In the face of random failures, it was found that average path length increases only 11% if no recovery algorithm is used to fix the network. CAN balances data load very well. This is because when a node is assigned a random space, it performs a 1-hop volume check to see if any neighbouring spaces are larger and should be split. With this scheme in place, approximately 82% of nodes have "even" share of the key space.

The author spends a considerable amount of time discussing network latency in CAN. Obviously, reducing path length and the number of physical network hops will reduce latency. The authors have implemented several methods of reducing path length. First, multiple coordinate spaces (realities) can be used. In this scheme, each node takes several pieces of the key space (two nodes may have the same part of the space in different realities). Thus, data can be replicated throughout the system for redundancy. As well, routing can be performed through different realities. Both these results reduce the number of hops to data. Second, multiple hash functions can be used to place replicas of data around the network. In this manner, a node would be responsible for two key spaces under two different hash functions. If one hash function does not yield data, the second one could be used. Third, proximity routing may be used to take into account factors such as physical distance between nodes instead of performing purely greedy routing. The authors have also experimented with proximity neighbour selection, zone sharing and geographic zone assignment. All these schemes are meant to increase data redundancy or reduce the number of hops from the searcher to the holder.

As the paper winds down, the author shows how to use CAN in order to perform multicast operations. As well, some other routing algorithms such as Tapestry, Pastry, and Chord are presented briefly and compared to CAN. The author believes CAN is a superior routing algorithm in DHT-based networks.

The author concludes with a discussion of some of the open research problems in and future directions for CAN. Questions include decreasing path length and number of neighbours required for routing, quantifying the time taken to recover from failures (and reducing that time), and finding ways to exploit network heterogeneity.

Novelty

Although CAN is yet another DHT-based network, its novelty comes from the improved routing algorithm. This algorithm is scalable, balances load, handles random failures, and experiences low latency.

Organization

In general, this paper was well organized. Introductory concepts were introduced before the routing algorithm was presented. However, there were some places where data structures were introduced only when they were needed by an algorithm being explained. I would have appreciated an overview of the complete system before its algorithms were explained.

Writing Style

This article was extremely well written. Concepts were explained thoroughly and at a level that I easily grasped. Background information was well explained and did not just refer the reader to another paper. There were very few grammar and spelling mistakes, removing any ambiguities that might have existed.

Content

The information contained in this paper was complete and quite concise. Background information was well explained and did not just refer the reader to another paper. However, there were a few questions that the author left unanswered:

-How is data transfer performed when splitting a node or joining two nodes? Does data simply disappear from the system, or is it transferred to the new or joined node?

-Would it not be more efficient to have a node keep the same zone each time it logs in? Would this reduce the amount of lost or transferred data?

-How does the system handle targeted attacks? Is it resilient due to the redundant paths, or does it fail?