Review of Chord:
A Scalable Peer-to-Peer Lookup Service for Internet Applications

by Stoica et al.
– Kevin Roundy –

March 21, 2007

1 Contributions of Chord

Distributed hash tables provide for easy location of data by lookup on a key in applications such as peer-to-peer file-sharing, content distribution systems, web caching, and dynamic name services, to name a few. Starting from the idea of consistent distributed hashing to assign data to nodes, Chord provides a completely decentralized mechanism that allows a node to find the receptacle for a data item in O(log N) time by tracking only O(log N) locations of other nodes.

These features of Chord make it a scalable, cost-effective, and fault-tolerant solution for peer-to-peer systems since Chord requires no centralized components such as DNS servers, making Chord-based peer-to-peer systems truly peer-to-peer. Chord can be used effectively for cooperative web caching, with data mapped to caches by consistent hashing functions and Chord used to find the cache that contains the desired web data, with its URI as the key for the DHT.

2 Shortcomings of Chord

One might suspect that Chord would be widely used in file-sharing networks, but this is not currently the case. There are two main problems with organizing data based on distributed hash tables for such applications. First, the only type of lookup that is possible via Chord is exact key lookup. In order to support keyword search Chord would have to employ a separate mapping from keywords to DHT keys. Second, and more importantly, Chord requires files to be placed on nodes according to the DHT, which is not what file-sharing network users do, they make available some subset of the files that are already sitting on their hard drives.

Another problem with Chord is that while its load balancing ensures that no node will have to carry more than k/n items (k being the number of keys and n the number of nodes), it doesn't consider that some data items could be substantially larger than others. In the case of web caching, most cached data could be textual web data, interspersed with some large videos, causing problems for the nodes that have to host the videos in addition to the normal load of textual data items.

Replication for fault-tolerance is left up to the applications. This may actually be reasonable, and in keeping with the end-to-end argument, if the rate of failure is not particularly bad, which in many peer-to-peer scenarios, it will be. Application writers may not be aware of just how prone individual nodes can be to failure. For example, if a failure-prone node joins the peer-to-peer system and takes over an application's data, an application that has come to expect that its data is rarely lost can suddenly be disappointed. If replicated data is not read-only, however, it might be unreasonable to suggest that Chord be adapted to handle that issue.

3 Implications of Chord

Whenever it is possible to structure the distribution of data across a number of hosts in a network, it is worthwhile to consider using a distributed hash table for distributing the data objects and using Chord for finding the nodes that host them. The alternative to using Chord is to provide a more centralized mapping between keys and the nodes that contain them. This is likely to be more costly, and unless the centralized naming service is replicated it will be a single point of failure,

but it also provides its advantages, for one, it reduces the time required to find the node for a data item from O(log N) to O(log 1).