Critique

Problem Addressed

The paper aims at giving a protocol for efficiently locating the node storing a particular data item in a peer to peer system. It gives a protocol, which shall be used for getting the data item, and an application shall be built on top of this protocol, which shall decide which data item it wants to get, and what it wants to do with it. A few example applications are mentioned: Cooperative Mirroring, P2P sharing (torrent, kazaa).

Approach

Chord is a distributed hash table protocol, that specifies which keys should be associated with which node – taking care of load balance, and how an arbitrary node can search for an arbitrary key. It uses a technique similar to consistent hashing, but does not assume that each node has complete information. Each node stores information about only O(log(n)) other nodes. This thus results in a significantly smaller routing table than the traditional consistent hashing algorithm, making it more scalable.
The use of SHA1 hash to prove the complexity 'with high probability' was very clever, as was the maintainance of the fingers, which reduced the storage from O(n) (in consistent hashing) to O(log(n)), and also decreased the communication time from O(n) (when only successors are maintained) to O(log(n)).
An important feature of this protocol is that it is completely decentralized, i.e. no single node is more important than another one.
Another feature of this protocol is that it supports a resonable number of frequent arrival and departure of nodes. This makes it suitable for use in a real life application, like torrent, where users are logging in and logging out all the time.
The algorithms simplicity - thus reducing latency, provable correctness and provable performance - O(log(n)) for search and O(log2(n)) for adjustment - make it an attractive option for use.

Shortcomings and how they can be overcome

- It does not exploit network locality very well. For example, if a node i starts searching for a key that lies with a node very close to i – but anticlockwise – the protocol would go through almost a full circle in the clockwise direction before it reaches the destination. *This can be overcome by keeping two finger tables per node, one each for the clockwise and anticlockwise direction. The search can then be done by looking at increasing distances from the node in both directions alternately, like a breadth first search.*
- The protocol decides what keys(and thus the associated values) are associated with which nodes. This may not always be possible for the protocol to do in real life, since often the keys present on a node are fixed (e.g. The files shared by a machine using torrent). *This problem is solved if the data item being stored is not the key-value pair containing the file itself, but a list of nodes that contain that particular file. After obtaining the list, the application can then directly fetch the required file from the particular node.* Rapidly updating the lists of nodes once a node leaves the system, is still tricky.
- The order in which the nodes are scanned is deterministic and linear (one after the other). Hence the speed of access in the search stage is limited by the slowest node in the cycle of nodes accessed. This is not really desirable, especially in a system where the performance of various nodes is not known and can not be determined beforehand. *We could do multiple searches in parallel. For example, in the two-fingers-per-node solution discussed above, the node could do both clockwise and anticlockwise searches simultaneously.*
- If the protocol given in section 5 of the paper is used, one can never be sure if the Stabilization operation has occurred or not. Hence, if the nodes in the affected region

have incorrect successor pointers, there may occur **false negatives**. One can never be entirely sure if an element is actually not present, or if it is a false negative. False negatives may be unacceptable in some applications. *However, the paper assures that with high probability, this will not occur. One way to prevent it is to use the protocol in section 4. However, it does not let nodes be inserted and removed as easily. We must pay a little price in terms of surety of result, to get better complexities.*

- The paper does not give any way in which broken rings can be recombined. The future-work section gives a vague idea about how it might be done, but globally checking for the complete ring would defeat the decentralization advantage that we talked about earlier. *An approach known as strong stabilization has been given in the advanced version of this paper.*

Future Work

This paper was published nine years ago, and has been a pathbreaking one in its field, having been cited more than 8000 times. There have been several variations and extensions done to the algorithm already, such as incorporating privacy-preservation[4] and checking for formation of multiple cycles[5]. However, there are still many areas where this algorithm can be used.

- One areas where there has been a lot of advancement in the last several years is the advent of social networking sites. This algorithm can be used to incorporate file sharing within social networking sites. The users will thus not have to upload the files to the server, but can be shared in a P2P manner. This makes sharing of large files possible without burdening the storage space of the server.
- This can also be used in the back-end of social networking sites, or anything where large distributed databases are used. Failure of nodes is a less common occurrence in this scenario, and thus the algorithm desribed in Section 4 can also be used.

References

1. http://en.wikipedia.org/wiki/Chord_(peer-to-peer)
2. http://en.wikipedia.org/wiki/Consistent_hashing
3. http://www.mpi-inf.mpg.de/departments/d5/teaching/ws04_05/Proseminar/2004-11-23_Paper1.ppt
4. Privacy-preserving indexing of documents on the network : Mayank Bawa, Roberto J. Bayardo, Rakesh Agrawal and Jaideep Vaidya
5. http://pdos.csail.mit.edu/chord/papers/