Review of the paper: “Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications by Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan”

Reviewer: Varun Kesharaju

I. Comprehension
   A. Distributed systems have many desirable features such as redundant storage, selection of nearby servers, search, hierarchical naming etc. They aim to remove bottlenecks or central points of failure from a system, allowing for better scalability and reliability.
   B. In decentralized distributed peer-to-peer systems, the major task is to efficiently locate data. Distributed hash tables can be used to route requests and locate required data but they also assume a global knowledge of the network, which is very inefficient to scale.
   C. This paper proposes Chord, a lookup protocol to solve the problem of efficient scalable lookup given a key. It also tackles issues like load balancing, decentralizing, availability and flexing naming.
   D. The chord protocol resembles a ring overlay topology which uses consistent hashing to distribute data uniformly across the peers in the network. It is robust even in case of node failures and joins.
   E. This protocol is proven to be scalable logarithmically with the number of nodes, producing a lookup time around $O(logN)$. It requires knowledge about only $O(logN)$ other nodes in the network.
   F. Chord protocol uses concepts such as finger tables, predecessor nodes and list of successor nodes which help in the powerful lookup process and request routing.
   G. Node’s IP address is hashed to place them on an *identifier circle*. Given a hashed key k, that key is assigned to the first node in the clockwise direction (successor node) whose

identifier is equal to or follows (the identifier of) k in the identifier space.

H. Paper discusses two kinds of approaches to locate the successor node of a key; concluding that the latter method is faster than the other.
    1. Iterative
    2. Recursive

I. To achieve high load balancing, it is proposed that each node would accommodate multiple virtual nodes instead of having one identifier per physical node.

II. Critique

A. Chord relies on periodic stabilization routines that have to be run in regular intervals to avoid timeouts and maintain accurate routing tables and an updated state of the network. I believe that such a regular update process adds on to the network overhead and traffic which inturn increases the round trip time for a query. Frequently joining and leaving nodes, either voluntarily or abruptly would enforce heavy network traffic by the updates from stabilization routines. This degrades performance in highly dynamic environments. The *fix_fingres* and *stabilize* methods are to be called iteratively, which adds latency during updates. In scenarios where all of the nodes corresponding to a node's finger table have left the network, there might be many timeouts when the node is trying to update each entry of its finger table. Only the successor and one finger table entry are stabilized for each call, so the expected interval between successive stabilizations of a given finger table is much longer than the average stabilization method call frequency. I believe lazy updates would be more robust than consistent periodic iterative update/stabilize calls in each node. I think event driven updates like rumor-mongering or other epidemic based algorithms might be more beneficial rather than a fixed-interval update routine.

B. When a node joins or leaves the network, the redistribution of data takes a lot of time in practice even if the volume of data is

uniformly distributed over the nodes. In highly dynamic environments, moving $O(1/N)$ of the data during each node failure might not be a step with low penalty. As the paper leaves redundancy onto the application using Chord, I feel that there would be strong correlation between the time required to redistribute data and the frequency of stabilization function calls.

C. Results produced to prove Chord's robustness and efficiency does not take the host network into consideration. There might be cases where routing the request to a physically nearby node is more desirable than a node that is geographically at a farther distance which introduces greater overhead, which finally increases lookup latency.

III. Synthesis

A. Node Clustering Techniques for Network Partition Recovery

1. Motivation and proposal:
    a) When a network partition occurs for a long period of time, even if nodes possess some sort of global awareness there is a possibility that the routing tables in each partition would stabilize by the time partitions are connected back, raising issues like duplicates keys.
    b) Example:
        (1) Suppose *key_X* was originally assigned to N2 (in Partition A).
        (2) If N2 is unreachable, N5 (in Partition B) may take over *key_X*.
        (3) When the partitions reconnect, both N2 and N5 claim to be responsible for *key_X*, leading to an inconsistency.
2. Proposal and contributions:
    a) To fix this problem I would suggest that instead of relying on the nodes to heal the network by retaining knowledge of each other, a supernode-based mechanism (where a few

selected nodes in each cluster attempt to reach out to other clusters) would be beneficial.

3. Methodology:
    a) In case of network partitions each cluster node elects a supernode responsible for maintaining metadata about the cluster.
    b) Each supernode attempts to reach other supernodes periodically. When network connectivity is restored, supernodes initiate a merge protocol by sharing their stored metadata.
    c) Nodes in each cluster update their finger tables based on the recovered metadata, enabling a smoother reintegration.