Decentralized systems are peer-to-peer systems. Creating a scalable protocol that enables lookup in a dynamic peer-to-peer system is the goal of this study. A hash function is used by Chord to associate keys with nodes. Due to distributed storage, the Chord protocol will communicate with other nodes. As a load balancer, Chord distributes keys among the keys in an equitable distribution. Because it is decentralized, no node is more significant than the others. Additionally, it offers availability, which implies that even when a system is constantly changing, it still works. The name conventions of the chord protocol are rather flexible. Fast-distributed hash function calculation is made possible by Chord. Cord offers consistent hashing; for instance, the hash of a node is determined by hashing the node's IP address, and the hash of the key identifier is determined by hashing the key. Each node has a predecessor and a successor, and it takes the shape of a circular ring structure. As an illustration, the chord ring with 10 nodes and 5 keys is displayed. Nodes can join and exit the network at any time thanks to consistent mapping. When a new node joins the network, its IP address is hashed, and the hashed key is then provided to any of the other nodes, who will examine the key and determine where the new node should go. Since every node has a successor and a predecessor, reaching any node is simple for them to do. They are completely knowledgeable with the network and its users. This method has the significant drawback of requiring a lot of information storage as the network expands, which may be quite challenging to manage. The drawback of direct search is this. We can go to any node in o(1), but we must save a lot of significant data. Since we must search over the complete list of nodes to identify the correct one, linear search is necessary. In order to have finger tables for the chord algorithm, a middle ground would be much preferable. The lookup table will be cut in half since the finger table will only preserve entries up to a certain point.

Therefore, when a key has to be located, it only needs to check its finger table to see whether it can jump to the closest place and give the key to that node, which will then continue the process until the target node is found. If not, it may just hop to the furthest site it can reach. Since finger tables have nlogn entries, we can search anything in nlogn using them.

When a new node joins, it must be aware of at least one other node in the network. The hashed value of the node's IP address is used to determine its position, and a number of stabilize and notify messages are sent to allow the new node, as well as its successor and predecessor nodes, to update their information appropriately. According to the investigations, there is a significant variation in the number of keys kept at each node.

Each node often stores more than one successor to ensure resilience. We retain a different set of finger tables and assess the query to determine the nearest neighbor from the finger tables in order to address this issue. We know that the average path length is typically 0.5 log n, although lookup latency can be fairly high. Multiple virtual nodes can be used to improve performance. The simplicity, accuracy, and demonstrable performance of chord, together with the fact that it continues to work effectively even when node information is only partially accurate, make chord a more appealing method for employment in the future.