**COMPSCI 590S  Systems for Data Science** **Fall 2017**

# Lecture 12: Hashing, Distributed Hash Tables, and Chord

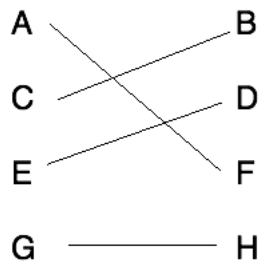
*Lecturer: Emery Berger* *Scribe(s): Samuel Ginzburg*

## 12.1 Quick Overview of HashTables

HashTables are a data structure that seek to provide O(1) access time on average to values given a specific key. HashTables use a hashing algorithm in order to hash the keys to an offset in an array, which can then be used to lookup the value associated with the key. Since arrays are of a fixed size, and HashTables grow in size over time, HashTables cannot be static, and must be dynamic.

The hashing function used to convert the keys into offsets for looking up the value in the array should generally form a random distribution if the algorithm is good.

Typically this results in a random projection from keys to values:



There is also the pigeonhole principle, which states that if a HashMap has M objects and N bins to place those objects in, and if M > N, then some bin has more than 1 item.

In addition to this we know that the expected length of the traversal is $E[\frac{M}{N}]$, where M is the number of objects, and N is the number of bins.

In addition, there are concurrent HashMaps, which allow for concurrent access by using locks.

There are implementations that use a singular global lock, and implementations that lock every single key with a lock, as well as implementations inbetween that form buckets and have a lock for each bucket.

Java Objects all have hashCodes via the hashCode() method derived from the Object class.

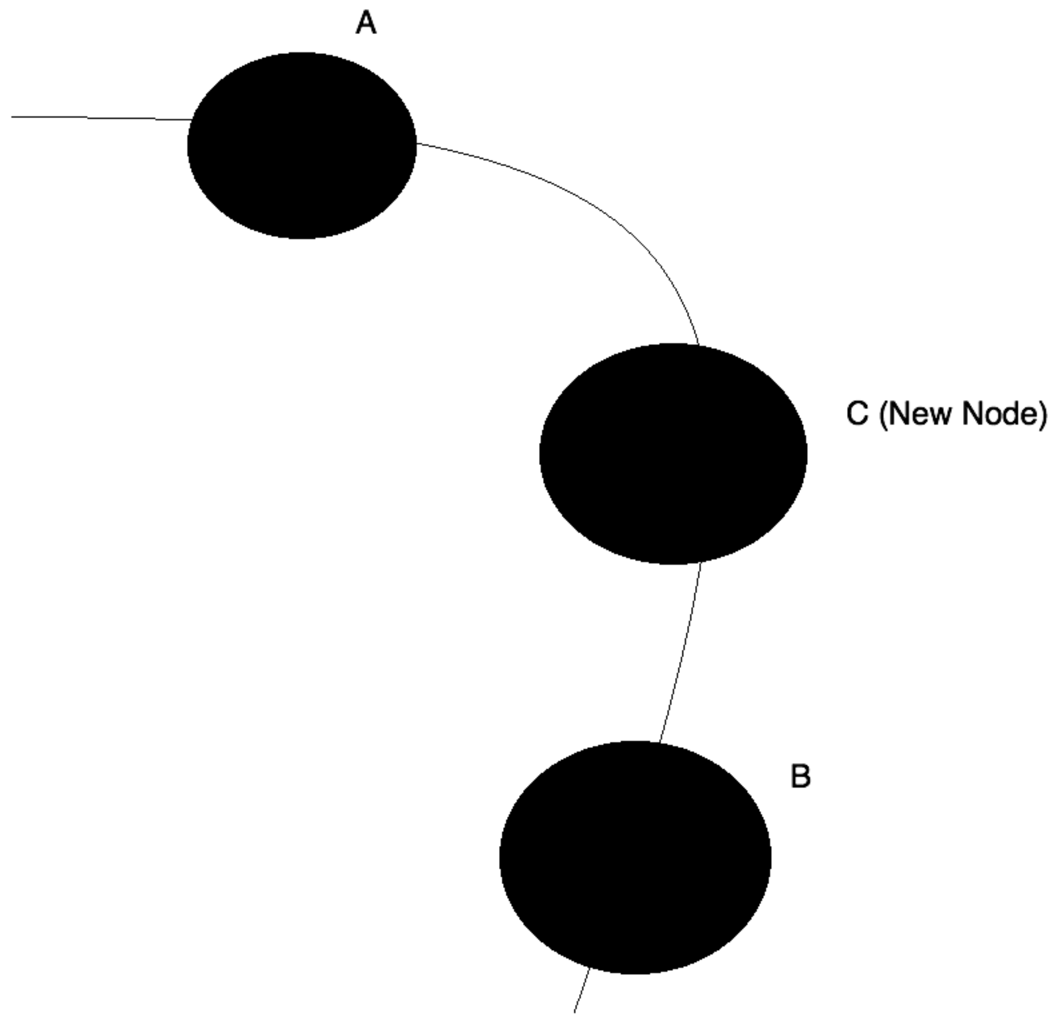
## 12.2 Distributed Hash Tables

The idea behind distributed hash tables is that the key-value pairs can be distributed onto multiple machines, to store particularly large hash tables.

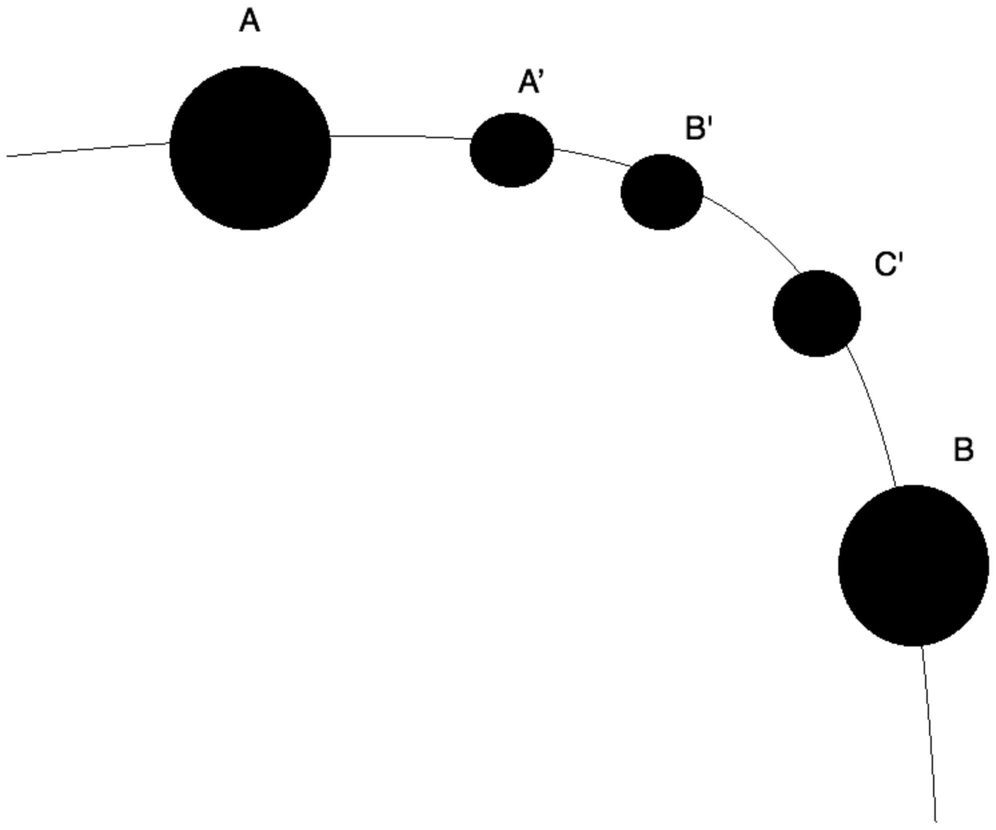
## 12.3 Chord

A problem with distributed hash tables is that once they get sufficiently large, if you have to split a bucket, you will have to rehash all the values in the hash table, which is an extremely expensive operation for sufficiently large data sets.

The Chord paper describes an algorithm for adding new nodes without having to rehash the values on every single machine.



In the diagram above we see that we were able to add a new node C into the chord system, and the only node that will have to be rehashed in the image above is B. This means that for each insertion of a new node, the number of rehashes is O(1), and not O(n), which would be computationally intractable for large key-value stores.



In between nodes in Chord, there exist virtual nodes, which point to an actual node. The purpose of these virtual nodes is to distribute load across the system. In the example above, we see a chord system with 3 actual nodes (A, B, C), and the virtual nodes for those three nodes are distributed evenly throughout the system like the image above.

## References

[1] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.