# MapReduce on a Chord Distributed Hash Table

Andrew Rosen  Brendan Benshoof  Robert W. Harrison
Anu G. Bourgeois

# Background

- Google's MapReduce [1] paradigm is integral to data processing.
- Popular platforms for MapReduce, such as Hadoop [2], are designed to be used in datacenters with a degree of centralization.
- Until recently, analysis and optimization of MapReduce has largely remained constrained to that context.

# Goals

- We wanted build a more abstract system for MapReduce.
- We remove core assumptions [3]:
  - The system is centralized.
  - Processing occurs in a static network.
- The resulting system must be:
  - Fault tolerant.
  - Scalable.
  - Completely decentralized.

# Features of ChordReduce

ChordReduce is a decentralized framework for distributed computing:

- Scalable.
- Load-Balancing.
- Decentralized:
  - No centralized node is needed to maintain metadata.
  - No central coordinator for tasks.
- Fault tolerant:
  - The loss of multiple nodes does not impact integrity.
  - The network can withstand numerous simultaneous faults.
  - Nodes in the network autonomously repair damage.

# System Architecture

ChordReduce has three layers

- Chord [4], which handles routing and lookup.
- The Cooperative File System (CFS) [5], which handles storage and data replication.
- MapReduce.

# Chord

Chord is a distributed hash table (DHT), where the nodes in the network are arranged in a ring overlay.

- Nodes and files are assigned a $m$-bit key.
- Chord gives a high probability $\log_2 N$ lookup time for any key.
- Nodes know their predecessors and successors in the ring.
- Nodes also maintain a table of $m$ shortcuts, called fingers.
- Nodes are responsible for files with keys between their predecessor's and theirs.

# A Chord Network

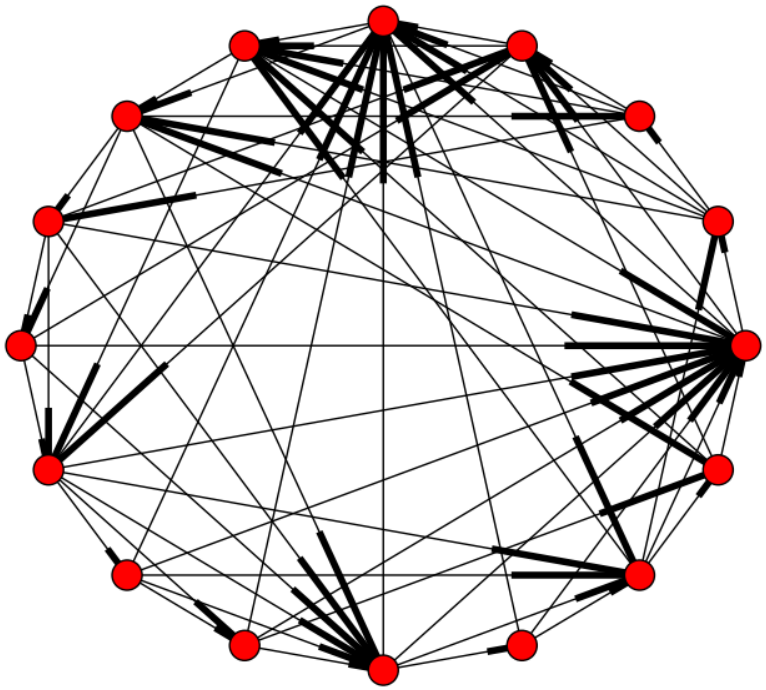

Figure: A Chord ring 16 nodes where $m = 4$.

# CFS

The Cooperative File System runs on top of Chord.

- Files are split up, each block given a key based on their contents.
- Each block is stored according to their key.
- The hashing process guarantees that the keys are distributed near evenly among nodes.
- A keyfile is created and stored where the whole file would have been found.
- To retrieve a file, the node gets the keyfile and sends a request for each block listed in the keyfile.

# Fault Tolerance

- Each node maintains a list of its $s$ closest successors.
- Nodes back up data they're responsible for to their successors.
- When a node's predecessor fails, the node can immediately take over.
- The network will only lose data if $s + 1$ successive nodes fail simultaneously.
- The chances of this are $r^{s+1}$, where $r$ is the failure rate.

# Starting a MapReduce Job

- Jobs can be started at an arbitrary node, denoted the *stager*.
- The stager retrieves the keyfile and sends a map task for each key.
- Once the
- Return address

# Data Flow

- Distributed tasks
- Return address

# Fault Tolerance

# Fault Tolerance

# Experiments

Stuff about Chord

# Summary of results

Stuff about Chord

Questions?


J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.


"Hadoop," http://hadoop.apache.org/.


"Virtual hadoop," http://wiki.apache.org/hadoop/Virtual


I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, August 2001. [Online]. Available: http://doi.acm.org/10.1145/964723.383071


F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.