MapReduce on a Chord Distributed Hash Table

Andrew Rosen Brendan Benshoof Robert W. Harrison Anu G. Bourgeois
Department of Computer Science
Georgia State University
Atlanta, Georgia
rosen@cs.gsu.edu

Google's MapReduce paradigm has rapidly become an integral part in the world of data processing. Popular platforms for MapReduce, such as Hadoop, are explicitly designed to be used in large datacenters. A consequence of this is the analysis and optimization of MapReduce has largely remained constrained to that context. We were motivated to create a MapReduce framework that could be easily deployed in other contexts, such as cloud computing or peer-to-peer networks, with minimal configuration. A more general-use framework for MapReduce cannot make the assumptions that other frameworks rely on, such as a high performance infrastructure, a static network, or a centralized source to organize and coordinate the network and its resources.

We present ChordReduce, our MapReduce framework. It is highly robust, scalable, and does not require a centralized source of coordination. ChordReduce is completely decentralized, avoiding the single points of failure that exist in Hadoop and other frameworks. Responsibilities that required a central coordinator, such as handling metadata, coordinating the execution of MapReduce, or assigning backups are all handled by the underlying Chord protocol.

ChordReduce is built on top of Chord, a peer-to-peer (P2P) networking protocol for distributed storage and file sharing that provides $\log(n)$ lookup time for any particular file or node. Files and nodes are evenly distributed across a ring overlay and organized so the responsibilities of a failed node are automatically reassigned. We leverage Chord's properties to distribute Map and Reduce tasks evenly among nodes and maintain a high degree of robustness during execution. The loss of a single node or a group of nodes during execution does not impact the soundness of the results and their tasks are automatically reassigned. An additional benefit is nodes which join the ring during runtime can automatically have work distributed to them.

We implemented ChordReduce by building our own version of Chord plus a module to run MapReduce operations over the network. Files are split into blocks and stored on the network using our implementation of CFS, a block file system. File splitting can be user defined or handled automatically by ChordReduce. When the user wants to run a MapReduce job over a file or files, a task is sent to each node storing a block of that file. In the case of computations that are purely mathematical, tasks are uniformly distributed across the network.

Our experiments establish that ChordReduce operates even under extremely high rates of churn and follows the desired logarithmic speedup. These results establish that ChordReduce is an efficient implementation of MapReduce. The applications are far-reaching, especially for big data problems and those that are massively parallel. Implementing MapReduce on a Chord peer-to-peer network demonstrates that the Chord network is an excellent platform for distributed and concurrent programming in cloud and loosely coupled environments.