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

***Abstract*—The abstract should be no more than four sentences long here I think.**

***Index Terms*—MapReduce; P2P; Parallel Processing; Peer-to-Peer Computing; Cloud Computing; Middleware;**

I. Introduction

Google's MapReduce [1] paradigm has rapidly become an integral part in the world of data processing and is capable of efficiently executing numerous Big Data programming and data-reduction tasks. Using MapReduce, a problem can be split it into small, functionally identical tasks whose results can be combined into a single answer. MapReduce has proven to be an extremely powerful and versatile tool, providing the framework for using distributed computing to solve a wide variety of problems, *such as sorting, reverse indexing (citation)*.

II. Proposed approach

Follows up on the closing of the intro. We use Chord as the backbone to mapreduce. This is a general overview

Our work concerns itself with the design and implementation of ChordReduce, a novel implementation of Chord that acts as middleware for creating and running MapReduce jobs. ChordReduce satisfies the desired properties for a distributed MapReduce platform. Chord is a peer-to-peer 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 such that the responsibilities of a failed node are automatically reassigned. ChordReduce leverages these features 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 that nodes joining the ring during runtime can automatically have work distributed to them.

MapReduce frameworks are generally hierarchical, with the responsibility of scheduling work, distributing data and tasks, and tracking progress at the top. This leads to centralized MapReduce implementations having a single point of failure. The ChordReduce framework distributes both responsibility and work among its members, eliminating the need for a central source of coordination. ChordReduce provides ideal characteristics for a MapReduce framework by being highly scalable, fault-tolerant during execution, able to handle a high degree of churn, and minimizes the amount of traffic that results from maintaining the network.

III. Implementation

Talk about actual implementation and collection of the data

MapReduce frameworks have two components: the MapReduce data processing component and a distributed file system, such as the Google File System [2] or the Haddop Distributed File System (HDFS) [3]. The file system

We implemented ChordReduce by building our own version of Chord plus a module to run MapReduce operations over the network. For our experiments, we wanted to establish that the speedup from distributing the work would follow an expected logarithmic pattern, even under differing rates of churn. To this end, we wrote a MapReduce job that would approximate the value of π. Our experiments varied the rate of churn the network experienced as well as the degree of desired precision, which directly correlated to the number of jobs distributed among the nodes.

Our results showed that ChordReduce operates even under extremely high rates of churn and followed 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.

IV. Preliminary Results and Conclusions

References

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

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[3] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design," 2007.