MapReduce on a Chord Distributed Hash Table

Abstract

MapReduce platforms are designed for datacenters, which are highly centralized in nature. These platforms require a centralized source to maintain and coordinate the network, which leads to a single point of failure. We present ChordReduce, a generalized and completely decentralized MapReduce framework which utilizes the peer-to-peer protocol Chord. ChordReduce's robustness, scalability, and lack of a single point of failure allows MapReduce to be easily deployed in a greater variety of contexts, including cloud and loosely coupled environments.

Introduction

Google's MapReduce (Dean and Ghemawat, 2008) 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

ChordReduce is built on top of Chord (Stoica et al., 2001), 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.

MapReduce frameworks are generally hierarchical, with a centralized source responsible for scheduling work, creating and placing backups of data, or the status of nodes. This leads centralized MapReduce implementations to having a single point of failure.

Our framework allows us not only to distribute the data and tasks among members of the network, but distributes the responsibility of maintaining the network as well, removing the need for any centralized point of failure from the network. ChordReduce provides ideal characteristics for a MapReduce framework by being highly scalable, fault-tolerant even during execution, and minimizes

the work need to maintain the network. The lack of a single point of failure allows ChordReduce to be deployed in a greater variety of environments and contexts, ranging from datacenters, P2P networks, or cloud computing.

Implementation and Experiments

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 (Dabek et al., 2001). 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.

Table 1: The results of calculating π generating 10^8 samples under churn. Churn is the chance for each node to join or leave the network. The large speedup is from joining nodes acquiring work during experimental runtime.

Churn per second	Average runtime (s)	Speedup vs 0%
0%	441.57	1.00
0.00250%	331.80	1.24
0.00775%	445.47	0.92
0.025%	431.86	0.95
0.4%	329.20	1.25
0.8%	191.25	2.15

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. We implemented two MapReduce jobs for experiments: a Monte-Carlo approximation of π and creating a word frequency list on a document. Our experiments varied the rate of churn the network experienced, the number of workers in the network, the number of tasks distributed among the nodes, and the frequency at which maintenance occurred. Table 1 shows the experimental results for calculating π different rates of churn in an initial ring of 40 deployed nodes. Our results showed that performance did not suffer under even very high rates of churn and that the reassignment of work at runtime to joining nodes allowed the network to perform better in some cases.

Conclusion

Our other experiments 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.

References

Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-Area Cooperative Storage with CFS. *ACM SIGOPS Operating Systems Review*, 35(5):202–215, 2001.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *SIGCOMM Comput. Commun. Rev.*, 31:149–160, August 2001. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/964723.383071. URL `http://doi.acm.org/10.1145/964723.383071`.