Research Philosophy

Andrew Rosen

1 Executive Summary

I research and develop fault-tolerant systems because they are absolutely critical to the Internet and networking, which have become an essential part of our lives. Fault-tolerant and robust systems are what hold a complex environment like the Internet together. The Internet is made up of billions of devices, each the source of thousands of tiny errors every day. The packets are dropped, the routers are poisoned, the hard drives keep crashing, someone tripped over a power cable, the software is full of bugs, the developers meant to add security but it was adopted before we got the chance, and nobody can connect to the printer. The Internet, by all rights, should not work, but it does because smart people decided that failure was an assumption that needed to be baked into every layer of every protocol.

Fault-tolerance is in many ways the most important part of what makes the Internet and networking work in general. As companies grow and create larger and larger intranets and datacenters, as more people gain access to the Intenet, adding billions of handheld devices that rival the power of last decade's consumer desktop, this problem is only going to get magnified.

1.1 Broad Overview

My research primarily focuses on Distributed Hash Tables (DHTs), with applications towards peer-to-peer (P2P) networks, Big Data, and Distributed Computing. DHTs are primarily used in P2P applications due to their decentralization and robustness . These qualities make them well suited for applications such as file-sharing, content distribution, multi-player video games, botnets, and video chat. Their qualities also make them suited for distributed computing.

My research on distributed computing over distributed hash table has yielded a platform I have worked with researchers not just in Computer Science, but Psychology, Astronomy, and Biology as well. I have seen first hand how fields outside Computer Science draw more and more heavily from distributed computing and big data. My platform can be used to solve not just problems in Computer Science, but be used by researchers in other fields to quickly create and deploy systems for solving Big Data problems.

I like things that keep working when they break.

Regex search github for passwords

2 Past and Future Work

My past and current research focuses on using DHTs for distributed computing [4] and defining a DHT at its most abstract level. There are many different types of DHTs, but all DHTs share very important qualities. They are *scalable*, which means that each additional node in the network minimally impacts the cost of keeping the network organized. DHTs are also highly *fault-tolerant*. Unlike many other systems, DHTs assume that nodes will be continuously entering and leaving the network. Because of the way DHTs are organized, they can handle large scale failures, such as a power outage affecting an entire city. The last quality of DHTs is that they are *load-balancing*. This means the data stored in a DHT is evenly distributed among nodes in the network using a cryptographic hash function.

2.1 Distributed Computing on A DHT

The scalability, fault-tolerance, and load-balancing qualities of DHTs are also highly desirable qualities in a distributed computing framework. As such, I wanted to see if it was possible to use DHTs to perform

distributed computing.

ChordReduce [4] was a proof of concept for using distributed hash tables for distributed computing. It used Chord [6], a well studied DHT, and exploited its various features to perform MapReduce [3] tasks, a very popular method for framing distributed computing problems.

Unlike other distributed computing frameworks, we made no assumptions as to the context in which ChordReduce would be used. ChordReduce could be used in either a large datacenter or in completely heterogeneous, peer-to-peer context. In a heterogeneous network, we assume all the nodes represent different pieces of hardware, with differing levels of computational power and reliability. As a result, we needed to rigorously test the fault-tolerance of the system and made an exciting discovery while doing so.

To test fault tolerance, each node essentially flipped a coin weighted in its favor, and when it lost the flip, left the network and reentered as a new node. This simulated churn, turbulence in the network caused by the continuous entering and leaving the network.

Churn is normally a chaotic influence on the network. However, we found that at high levels of churn, nodes advantageously redistributed work in such a way that tests with high levels of churn performed better than tests with low levels of churn.

ChordReduce established three important ramifications. First, we experimentally demonstrated that DHTs are capable of being used as a framework for distributed computing. Second, DHT based distributed computing can have nodes enter the network at any time, even when a job is running, and be immediately put to work. Finally, we discovered that while churn is normally a disruptive force, at high levels it can be helpful to the network.

2.2 Abstracting DHTs

I became interested in finding if there was a way of abstracting DHTs. While there has been extensive literature defining what a DHT does, none of these rigorously define how the various pieces of a DHT can be defined. For example, each node in a DHT has a range of keys which it is responsible for, but this range is defined differently for every DHT.

Working with my coauthor, I found that DHTs can be cleanly mapped to Voronoi Tessellations and Delaunay Triangulations. By treating each node as a Voronoi generator, a node's range of keys becomes it's Voronoi region and links to nearby nodes correspond with the links forming the Delaunay Triangulations.

We created a distributed greedy heuristic with which each node can quickly construct its own Delaunay Triangulations and Voronoi Tessellation [2]. Our heuristic works in any geometric space where the distance is defined.

We used this to create an abstract DHT called UrDHT [5]. UrDHT can be used to implement the topology and functionality of any other DHT by defining the space the DHT operates in.

UrDHT essentially acts as a "fill-in-the-blanks" for creating distributed hash tables. The novelty of UrDHT is that it makes it easy for scientists in any field to construct a DHT based application. For computer scientists, UrDHT provides a means of rigorously comparing different architectures.

2.3 Dissertation

My dissertation focuses on implementing the distributed computing framework on a wider variety of topologies using UrDHT and creating an autonomous load-balancing strategy for nodes within a DHT. When I initially created ChordReduce, it used a single DHT, Chord [6]. One of the questions we needed to examine for future research was "did the choice of DHT topology matter, and to what extent?" UrDHT provides an excellent means for answering this question, since it UrDHT makes it easy to construct the topology of other DHTs.

The larger and more interesting question I am asking is how can churn be leveraged to speed up computation. I found in previous research that extremely high levels of random churn helped the node complete computation faster. This is because the ranges of keys nodes are responsible for are unevenly sized, which means some nodes end up with more work than other nodes.

Because of the rigorous fault-tolerance we built into the system, one of these nodes going offline was only minor inconvenience.

3 Future research

Our research would have many applications within the field of Computer Science. Traditional MapReduce frameworks and other distributed computing solutions are limited to being deployed solely in a datacenter [1]. Distributed computing frameworks built with UrDHT would be able to be deployed any context, be it a data center, a P2P network, or a volunteer computing pool. Furthermore, the autonomous load-balancing feature would allow frameworks to automatically give nodes with more power more work.

Our framework would not only be of interest to computer scientists. The "plug-and-play" nature we envision would make it easy for experts in other fields to use our framework to solve computationally intensive problems. Some example problems are Monte-Carlo Methods (of interest to mathematicians and economists) and machine learning (of interest to biologists).

This project would allow both organizations with large amounts of computing power and average developers with fewer resources to spend less time setting up and configuring their hardware to work together. The goal is for our software to make distributed computing more of matter of "plug-and-play," allowing researchers to spend less time setting up and maintaining computing platforms.

References

[1] Virtual hadoop. http://wiki.apache.org/hadoop/Virtual

[2] Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, and Robert W Harrison. A distributed greedy heuristic for computing voronoi tessellations with applications towards peer-to-peer networks. In *Dependable Parallel, Distributed and Network-Centric Systems, 20th IEEE Workshop on*.

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

[4] Andrew Rosen, Brendan Benshoof, Robert W Harrison, and Anu G. Bourgeois. Mapreduce on a chord distributed hash table. In *2nd International IBM Cloud Academy Conference*.

[5] Andrew Rosen, Brendan Benshoof, Robert W Harrison, and Anu G. Bourgeois. Urdht. `https://github.com/UrDHT/`.

[6] 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.