Research Philosophy

Andrew Rosen

1 Executive Summary

My research area focuses on developing fault-tolerant systems because they are absolutely critical to networking and the Internet, 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.

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.

2 Past and Current Work

My past and current research focuses on using DHTs for distributed computing [7] 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 [7] was a proof of concept for using distributed hash tables for distributed computing. It used Chord [9], a well studied DHT, and exploited its various features to perform MapReduce [5] 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 UrDHT

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 [1]. Our heuristic works in any geometric space where the distance is defined.

We used this to create an abstract DHT called UrDHT [8]. 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.

I am currently 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 [9]. 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.

2.3 Autonomous Load Balancing

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. As the computation continues, however, more and more nodes are idle without any work. When one of these nodes leaves the network and rejoins, it either will join in a location where it cannot acquire any additional work, or it will reenter at a location where it can. The former case has no impact on the network's running time, whereas the latter redistributes some of the work. This was the source of our speedup in our experiments.

If an essentially random application of churn can result in a speedup, is there a strategy I can use to control this process and create a larger benefit to runtime? I am currently examining multiple strategies that would leverage the "benefits" of churn. These strategies would allow nodes to strategically and autonomously redeploy themselves to new locations when needed, which would mean workloads could be redistributed on the fly. To address differences in computing power in a heterogeneous network, computationally stronger nodes could represent themselves as multiple nodes within the network. This is effectively applying the Sybil attack [6], but in a controlled manner to the network's benefit.

3 Future Research

Wireless Sensor Networks Sleep Strategies Cărbunar et al. [2] demonstrated how Voronoi tessellations could be used to solve the *coverage-boundary* problem in wireless ad-hoc networks. The coverage-boundary problem asks which nodes are on the physical edge of the network. This problem has applications such as setting sleep schedules [3]. Since our greedy Voronoi heuristic is designed for a distributed environment, it appears well suited for applications in WANETs.

Interdisciplinary Applications UrDHT has many applications for interdisciplinary research, as it makes it easy to deploy for distributed problems that occur in a wide variety of disciplines. Monte-Carlo approximations are particularly well suited for UrDHT's computation model.

Another set of wide-ranging problems is classification. I can design the system to build a classification model from test samples and distribute this model to the nodes containing data to be classified. Some examples problem this can be applied to is classifying supernovae, or evaluating random forests.

Content Distribution Networks

Distributed DNS The Stop Online Piracy Act and Protect IP Act, both introduced and vigorously debated in 2011, brought attention to the fact that DNS could be fractured by filtering by the US government. This is a consequence of DNS being centralized and adopted before security was fully addressed. One of my goals is to deploy a fully decentralized and distributed DNS servers using UrDHT. A decentralized DNS service was previously considered impracticable [4], but new technology, such as the Blockchain, have addressed a number of previous concerns.

References

[1] 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.*

[2] Bogdan Carbunar, Ananth Grama, and Jan Vitek. Distributed and dynamic voronoi overlays for coverage detection and distributed hash tables in ad-hoc networks. In *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, pages 549–556. IEEE, 2004.

[3] Xinyu Chen, Michael R Lyu, and Ping Guo. Voronoi-based sleeping configuration in wireless sensor networks with location error. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pages 1459–1464. IEEE, 2008.

[4] Russ Cox, Athicha Muthitacharoen, and Robert T Morris. Serving dns using a peer-to-peer lookup service. In *Peer-to-Peer Systems*, pages 155–165. Springer, 2002.

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

[6] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[7] 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*.

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

[9] 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.