CS 244 Project 3

Sweet Song(shihui) and Jason Zhao(jlzhao)

Jellyfish - Networking Datacenter Randomly

Synopsis

This paper proposes a new network topology for data centers. Jellyfish uses a random graph to create a high bandwidth and fault tolerant network for data centers compared to traditional methods such at fat trees. The main idea behind random topologies is that it has shorter paths between servers and also have multiple disjoint paths to increase bandwidth. The authors noticed that existing path finding algorithms such as ECMP is not suffient to reach maximum capacity and propose several new path finding algorithms to increase perform. In the end, the authors show that random graph topologies has similar bisection throughput compared to traditional fat trees and is more extensible when adding new servers.

Results to Reproduce

The most interesting aspect of the paper is showing that a datacenter running using random graph topology can achieve the same actual bisection bandwidth as traditional fat tree topologies. We plan on replicating the results in figure 9 and figure 11 and hope to show that Jellyfish scales with fat tree as the number of servers in the data center increase. The authors used simulators to compute the expected throughput and we plan on using Mininet to achieve the same result. In addition to replicating the authors results, we aim to examine if there are any improvements to the path finding algorithm used by the authors. The conclusions from the authors were to use k-shortest path with Multi-Path TCP, we want to see if there is any other protocol or algorithm that will leverage the random topology more efficiently.

Testing Methodology

We ran our experiment on a Fedora 18 quad core Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz with 7G memory. It took a total of 5 minutes to finish the experiment when running the bash script in its entirety. The bash script includes calls to our python files to generate the Jellyfish topology, find shortest paths, and finally graph the results. Our topology mimics the topology as stipulated in Figure 9 of the paper. We have 686 servers with 12 ports each for a total of 2520 links. The python script *genGraph.py* generates this topology uniformly randomly to create the result of g_210_12.txt. Then $traffix.py$ outputs 687 pairs of nodes to "traffic.txt" to calculate the shortest paths between. Next, "genPaths.py" reads in both text files to calculate the $k = 9$ shortest paths, 64-way ECMP, and 8-way ECMP for each. Interestingly, it that the y-axis of the graph

We will run our experiment on datacenters containing between 1 and 5000 servers at some fixed interval. We will use the authors random sampling algorithm to generate the topology and use

Mininet to emulate the data center. In addition, we will implement our own switch logic and end host logic to support multi-path tcp and k-shortest path routing. We will also implement a fat-tree topology using the same number of servers and compare the aggregate througput of the 2 networks. We will test both topologies using both a uniform workload and data shuffle workload similar to traffic seen in Map Reduce applications.