Live Incremental Expansion of Clos Topology with Minimal Rewiring

By Daniel Chafamo, Jonathan Kosgei, Sancharz Gore

Abstract

This project is based on Google's experiment of the same title. Clos network topologies are the most commonly used topologies because they can support large scale data center networks with commodity switches. Use of commodity switches in a large data center network is cost-effective because commodity switches are cheap. In addition Clos network topologies are failure-resilient because they have many alternative paths between each source-destination pair that can be used in the event of a link/switch failure. However, expanding Clos networks can be a challenge because the amount of rewiring needed to incorporate new switches while maintaining Clos-like properties can be large, requiring substantial reduction in network flow during expansion. Our project aims to reduce the number of changes that need to be applied to the network to add a new switch in order to make expansion faster and less costly to the network flow. We use Integer Linear Programming(ILP) to generate a minimal rewiring sequence while maintaining Clos-like properties and install the changes on a like network.

Introduction

Data center topologies fill up with servers and storage gradually and it takes roughly 1-2 years for them to reach almost-full capacity. Therefore, initially building a large data center topology leaves substantial capacity idle for an unreasonable amount of time, which is very costly. Presently, the approach that is taken is that a moderate-scale network topology is built first. This topology is continually expanded just ahead of server arrival while the network is carrying live traffic. The objective of our project is to incrementally expand a Clos network topology and keeping rewiring to a minimum during this expansion.The goal was to expand the network topology live, without taking it out of service. It is unnecessarily disruptive and expensive to take the network out of service during expansion. Expanding the network by taking it out service will not only negatively affect availability of computing and storage capacity, but will also require the migration of applications using that network. Therefore, during the data center network expansion, the network should be available and minimal packet loss is expected to occur. The expansion ought to be fine grained.

The need for the network to be live during expansion requires that the expansion be fine-grained and be done incrementally in small stages. Fine grained expansion is expansion that is carried out at the server block of the topology. The server block of the topology consists of the aggregate switches only (as shown in image 1 below).

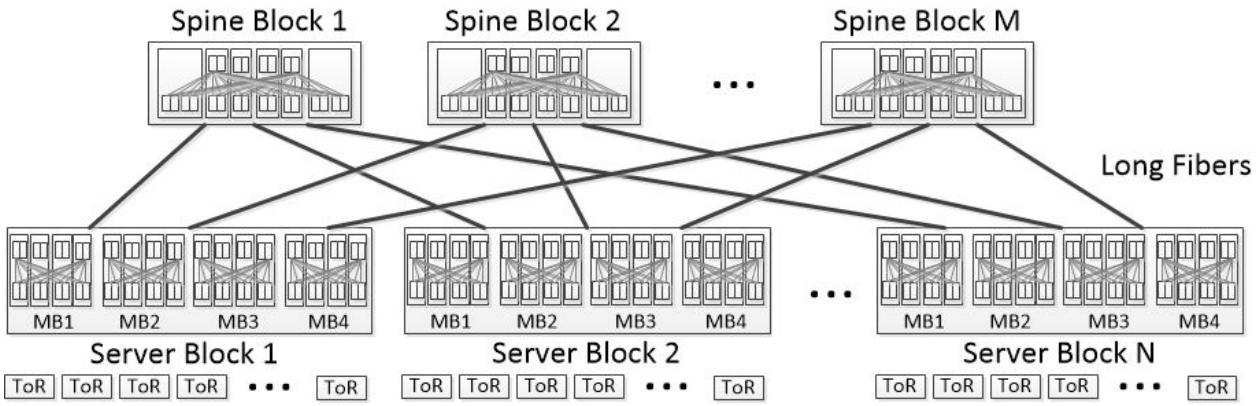


Image 1

Fine-grained expansion is preferred over coarse-grained expansion because coarse-grained expansion deploys a large amount of capacity at once, which is resource-inefficient. Fine-grained expansion is carried out in multiple automated stages. Each stage disconnects and adds a limited subset of network elements (to guarantee connectivity). Each stage has to be completed in minimal time to increase efficiency of the overall expansion process. Many links are rewired at each stage, but the goal is to minimize the rewiring that occurs during expansion. The fine-grained expansion is guided by an algorithm designed by Google called Minimal Rewiring Solver for fine-grained expansion of Clos. This Minimum Rewiring Solver reduces the average expansion stages by 3.1 times.

Design

The Minimum Rewiring Solver

The Minimum Rewiring Solver is the algorithm deployed for the expansion. At a high-level, the minimum rewiring solver for clos topology expansion considers the pre-existing topology and designs a new larger one incorporating the additional switches. The minimum

rewiring solver uses Integer Linear Programming with balance related constraints to minimize the number or rewirings. The goal of the Minimum Rewiring Solver is to ensure that the resulting topology has high capacity and high failure resiliency.

The variables in the ILP formulation were:

Number of links between each pair of server block and spine block

$X_{i,j}$: {0, 1, 2, 3,...}

The objective function of the Integer Linear Programming formulation was:

$$min \sum |X_{i,j} - Y_{i,j}|$$

where $Y_{i,j}$ is number of links in the reference topology

Given an initial wiring, the goal is to find a new wiring that incorporates the new block (switch + associated links) and minimizes the changes that need to be made.

The constraints of the Integer Linear Programming Formulation were:

Number of links: $X_{i,j}$ *variables must be greater or equal to zero*

Number of ports: *we have a fixed number of ports for each server block(aggregate switches)*

Capacity: *ensure that uplinks of the server block (aggregate switches)are evenly distributed across spine blocks (core switches) to preserve clos-like properties (scaling out, redundancy,...)*

Absolute value constraint: *a link change should register as a positive cost - use linear representation of absolute value function:*

$$(X_{i,j}-Y_{i,j}) \leq X'$$
$$(X_{i,j} - Y_{i,j}) \leq X'$$

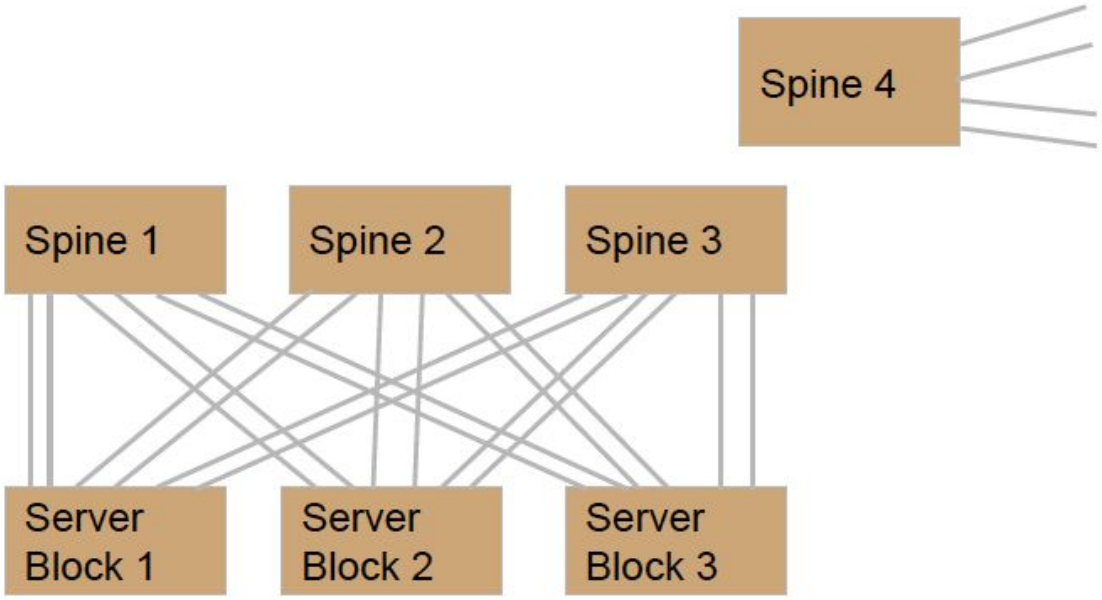
The Integer Linear Programming of the Minimum Rewiring Solver produces the most efficient command sequence output. A command sequence output is a series of instructions specifying

what switches to connect and disconnect in a prescribed order. The command sequence output connects and disconnects switches to incorporate the new spine block (core switches) being added to the topology. Since the initial routing instructions does not take into account the recently added spine block/core switch, after every few instructions, the topology is rerouted to incorporate the new spine block and newly added links.

Example Implementation

We started with an initial topology in Mininet and routing rules specified in the Ryu Controller (similar to Project B). We implemented a Equal Cost Multi Path routing strategy between the switches. The controller has an "Add Switch" functionality to support the Clos topology expansion.

Given a Clos Topology with 3 spine blocks (core level) and 3 server blocks (aggregate level), add a fourth spine block to this topology. Before addition of the 4th spine block, the topology has ECMP routing specified for the 3 server blocks and the 3 spine blocks.The diagram showing the initial topology configuration is shown in Image 2 below



The Minimum Rewiring Solver considers this initial topology and then uses to Integer Linear Program to produce a command sequence output that minimizes the number of rewirings and

ensures that the network is not disrupted.

The step-by-step command sequence output for the Integer Linear Programming formulation would b in the following format : (COMMAND, SERVER BLOCK, SPINE BLOCK)

('DISCONNECT' , 1, 1)

('CONNECT', 1,4)

.....REROUTE.....

('DISCONNECT', 2, 1)

('CONNECT', 2, 4)

.....REROUTE……..

('DISCONNECT, 3, 1)

('CONNECT', 3, 4)

.....REROUTE.....

After every connect-disconnect commands, the controller reroutes the topology to ensure that the new changes are incorporated.

Related Work and Future Work

Work related to this project includes the project that was conducted at Hebrew University of Jerusalem on Explicitly Expanding Expanders. The main objective of this project was to construct an an infinite sequence of expanders $G_0, G_1, \ldots,$ such that for every two consecutive graphs G_i and G_{i+1} , G_{i+1} can be obtained from G_i by adding a single vertex and inserting/removing a small number of edges, which we call the expansion cost of transitioning from G_i to G_{i+1} . This is very similar to the problem we were trying to solve in this project where

G corresponds to the Clos topology and adding a single vertex is fine-grained expansion at the server block level.

Further work that can be done in the project includes further maximizing available bandwidth during fine grained topology expansion to ensure that the network is not interrupted. In addition, improving visualization of traffic in the Data Center topology by incorporating functionality that enables users to graphically add aggregate switches/core switches on the User Interface. The visualisation can also be improved to show the network changes and how traffic is being routed in real time.

Conclusion

In this project we solved the problem of efficiently expanding a Clos topology. We utilized a Minimum Rewiring Solver to carry out fine-grained expansion at the server block.