Condor: Better Topologies Through Declarative Design

Paper Notes
Junzhi Gong

September 22, 2018

1 Backgrounds

- During the design of a datacenter topology, a network architect must balance operational goals with technical, physical and economic constraints, including being resilient to any failure, fulfilling every applications requirements, and remaining under budget, which are at odds with one another.
- Today, the process of designing topologies is decidedly manual. An architect typically begins by evaluating the utility and feasibility of topologies, and then evaluates minor variations of the topology.

2 Problems

- This tedious process of modification and evaluation in designing topologies severely limits the number of topologies that can be explored, leading to heavy reliance on intuition.
- Network operators may need to perform topology configuration on live networks, without reducing capacity below the networks defined Service Level Objective (SLO) during any operation.

3 Target

This paper describes *Condor*, a pipeline that facilitates the rapid generation and evaluation of candidate topologies, enabling an architect to aggressively explore the topology design space. It aims to make architects to explore a complex design space easily. It also helps architects to express, analyze, and refine specific topologies, while understanding the impact of design choices on tradeoffs between metrics such as bandwidth, fault tolerance, and routing convergence.

4 Solution

- **Fabric design factors.** This paper mainly focuses on the following design factors for network topologies. 1) Capital, installation, and energy costs. 2) Bandwidth and latency objectives. 3) Conceptual complexity. 4) Reliability. 5) Routing convergence time. 6) Expandability.

- **Topology description language.** This paper proposes a human-readable language for architects to describe high-level constraints on the topologies, named topology description language (TDL). TDL is a declarative language that is embedded in an object-oriented language. TDL helps architects to express a topologys structural building blocks and the relationships between them, along with potential connectivity, and constraints on connectivity.

  - ***Building blocks.*** An architect begins by describing a topology's common building blocks, each representing a physical component, such as a switch, patch panel, or rack, or a logical component, such as a "pod" in a fat-tree topology. The architect can also describe parent-child ("contains") relationships between building blocks.
  - ***Connectivity.*** Given a set of building blocks, the architect describes how they should be connected, by defining candidate connections with *connectivity pairs*, and then using *constraints* together with *tiebreakers* to eliminate potential connections.
    - *Constraints.* Constraints are associated with a TDLConnector object, and constrain the set of candidate connections in the connectivity pair's scope. Constraints applied to an individual building block can set the minimum, maximum, or exact number of connections for every component instantiated from that building block. Constraints applied to a pair of building blocks bound the number of connections for every pair of associated compo- nents, or between a single component of the LHS descriptor and all components of the RHS descriptor.
    - *Tiebreakers.* Tiebreakers are used to let architects to control the ordering of iteration of finding things to connect. The ordering is the key to finding a solution efficiently.

- **Network synthesis.** Synthesis of connectivity involves solving three problems: formulating synthesis as a constraint satisfaction problem; allowing an architect to influence the results via tiebreakers; and optimizing the synthesizer's performance to support a rapid design cycle.

  - ***Synthesizing the hierarchy graph.*** The synthesizer constructs a hierarchy graph, beginning by converting the root building block instance into the graph's root "component", and then recursively converting contained building blocks (expressed with the "Contains" relationship in TDL) into successor components. Once the synthesizer has instantiated all components, it generates groups based on the grouping rules defined in the TDL.

– ***Synthesizing the connectivity.*** Components are processed in reverse of the order in which they were instantiated. Thus, connectivity generation begins at the lowest non-leaf level or stage in the hierarchy graph. It is a CSP (constraint satisfaction problem).