Building Application-Aware Networks Infrastructure with Software Defined Networks

ABSTRACT

1. INTRODUCTION

2. MOTIVATING EXAMPLES

In this section, we charaterize the workloads running in today's datacenter, highlighting the features influencing the design decisions of A^2Net.

2.1 Shuffle

Uneven Load

2.2 Partition-Aggregator

Partition-Aggregator [1] workloads are usually generated by the data center applications handling user requests, e.g. emails, search engine and social networks content composition, which are usually built with soft real-time style for interactive. As shown in Figure 1, requests from users are split into multiple *sub-requests* by an aggregator in the root layer and delivered to the distributed worker processes in the system via aggregators on different layers.

To meet the all-up SLA [1], aggregators usually assign deadlines to the worker nodes in lower layers. When deadline arrives, the aggregator will directly return the collected results back to the upper layer instead of waiting for all nodes to complete. Those discarded results in lagged behind nodes can ultimately lower the quality of the responses. As we explained in Introduction section, network transfer occupies a considerable part of total workloads of data center applications. The worker deadlines mean that network flows carring requests and responses have deadlines. Only the network flows finished within deadlines can contribute to the throughput of application, on another side, the slow and ultimately discarded flows can waste network bandwidth.

However, the commonly used TCP protocol strives for maximizing network throughput while keeping fairness, being unaware of the deadlines of flows. The lack of prioritization among flows causes that *urgent* flows wait behind latency-insensitive flows just for fairness, making them miss the deadlines. To examine this issue, we reproduce the workload consisting of both

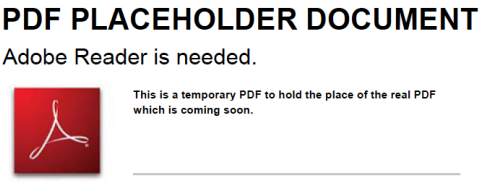

Figure 1:

throughput-preferred but latency-insensitive flows and latency flows....

3. TECHNICAL DESIGN

In this section, we will introduce the design principles of our system. We begin by discuss the stack of our system. Then examine the problems in current OpenFlow design which make it unsuitable for high performance networks and propose our solutions. Following that, we describe the the methods to maximize the throughput of latency-sensitve applications and reduce the straggler large flows

3.1 system stack

- API
- Flow Controller

3.1.1 challenges in openflow

1. involves controllers too often, those latency-sensitive flows will be suffered from it

3.1.2 solution

application table (deadline): static(intialized when warming up of switches)

dynamic table : job input information: priority, and flows input size

- for latency sensitive services applications, locating it in static one, operators should setup it manually, openflow controller sync the application information with the routers when warming up)

- for large ones, when invoking flows, the flow size is known by the controller through API, assigning rate according to input

4. REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.