Reducing Data Center Latency

Zainab Ghadiyali
zainab@cs.wisc.edu

Michael Griepentrog
mgriepentrog@cs.wisc.edu

Aditya Akella
akella@cs.wisc.edu

Abstract

Latency is usually compromised in order to improve bandwidth. However, with the advent of latency sensitive applications such as high frequency trading , the focus on reducing latency in network has increased. Furthermore, computation is increasingly also shifting to datacenters. Coupled together, these two factors necessitate further study on latency in datacenters. In this body of work, we study various factors that may influence latency, such as type of flows, offloading parameters, hardware such as network interface cards, optimized algorithms such as stochastic fair queueing and type of routers (presence and absence of switches). By utilizing a server-client architecture and OpenFlow switches, we show that offloading, stochastic fair queueing and utilization switches over direct flows help reduce latency in networks. Further, we quantify the difference and propose further experiments that would help identify other latency influencing factors.

1 Introduction

In the last few decades, the key focus of the network community has been on improving the overall goodput of networks. The initial focus on circuit switching moved to packet switching due to bandwidth and hardware inefficiencies in working with network resources for bursty communication traffic. The Transmission Control Protocol(TCP) was invented to address the issue of bandwidth/congestion collapse in the network and ensure bandwidth fairness. [1, 5, 7]
However network latency has been given less importance. The last 30 decades have brought a meager 30x reduction in latency, versus the 3000x times improvement in bandwidth. Several studies have been conducted in the 90s to tackle the latency issue, but they were not adopted. [3, 4, 8, 6] Additionally, most applications were throughput oriented (e.g. email) and hence not sensitive of delivery time. For example, switches are especially designed with large packet buffers in order to avoid congestion since TCP protocol does not deal well with packet drops from congestion. Network Interface Cards (NICs) are optimized to delay interrupts for upto 30 μs. Latency for latency sensitive applications is seen as an acceptable trade-off in order to maintain high bandwidth utilization. However, there is now an increasing interest in reducing latency in datacenters. A substantial amount of computing is now shifting to data centers and reducing latency is now easier given the confines of a building rather than tackling the issue for the internet at large. Furthermore, ultra low latency sensitive applications such as High Frequency Trading, HPC and Google Instant Search service could especially benefit from this. Several high-frequency traders implement trading applications with a goal to reach a trade decision in under 100 microseconds. NIC, end-host stacks and switches are all points where a packet may experience latency while traversing from client to server. In order to address delays arising from queueing delay at switches, DCTCP [1] uses ECN marking to slow down flows before the queue is saturated. HULL [2] further suggests trading off a little bandwidth to provide smaller amount of latency. In this paper, we look into understanding how switch, various offloading parameters and hardware influences latency.
OUR OBSERVATION IS THAT
THROUGH OUR EVALUATION WE FIND THAT

2 Methodology

2.1 Design

A strong evaluation and understanding of reducing latency will be incomplete without simultaeneously studying the trade-offs between latency and other important factors such as bandwidth and CPU utilization. Thus, we study latency along with its influence on CPU utilization and bandwidth. Furthermore, it would be important to also understand the role of latency influencing factors in UDP and TCP. UDP should be faster than TCP since it allows a continuous packet stream versus TCP that sends acknowledgements (ACKS) for a set of packets calculated using the TCP window size and round trip time (RTT) Thus, metrics were obtained for TCP and UDP flows. Offloading moves the IP and TCP processing to the NICs. Keeping type of NIC constant, we can study the influence of offloading on latency. TCP segmentation offload (TSO)/ Generic segmentation offload (GSO) are considered useful in increase outbound throughput of high-bandwidth network connections since it reduces host CPU cycles for protocol header processing and checksumming.Generic receiving offload (GRO) attempts to replicate the TSO modus operandi on the receiver-side. Effects of offloading is studied by:

1. TSO/GSO and GRO : On
2. TSO/GSO and GRO : Off
3. TSO/GSO : On and GRO : Off
4. TSO/GSO : Off and GRO : On

2.2 Experimental Setup

Two commodity computers connected via an OpenFlow switch [9]. One of the computers served as a client and the other as server. Eight experiments were ran in this setting. Another eight experiments ran on the two machines in absence of OpenFlow switch. For each experiment, throughput is recorded using iperf, cpu utilization through pidstat, interpacket delay through tcpdump and latency through ping. This process was automated by running bash scripts on server and client end. Offloading parameters were varied using ethtool. Data was analysed using wireshark and a python script that parsed results from experiments and provided basic summary in terms of min, max, stdev and average latency, average throughput in terms of Mbits/sec and average percent CPU utilization.

3 Evaluation

3.1 Bandwidth

TSO works by moving work of breaking down large chunks of data over to the NIC. In figavgBandwidthClientNoswitch, the x-axis represents offloading settings while y-axis represents bandwidth. Bandwidth is almost halved when TSO/GSO if switched off. This suggests that throughput may be increased by switching off offloading. Keeping GRO on in the absence of TSO/GSO did not show an appreciable change, thus suggested that GRO plays a little to no role in increasing bandwidth. Interestingly, the bandwidth remained fairly constant for switch as well as client despite varying offloading parameters when flows traversed through an OpenFlow switch.

In figavgcpuclientnoswitch, x-axis represents the offloading parameters, while y-axis respresents the average percent CPU utilization for TCP and UDP flows for a client when flows directly traverse from client to server. Irrespective of offloading settings, 100% CPU utilization is observed for UDP flows. This makes sense, since no offloading should occur for UDP flows. However, we see almost 100% CPU utilization for TCP flows in the absence of TSO/GSO. This is in line with findings expressed in figavgBandClientNoswitch. Since the NIC non longer performs offloading, CPU is utilized for segmentation. The CPU utilization increases 10x with a 2x decrease in bandwidth. In figavgcpuclientswitch, x-axis represents the offloading parameters, while y-axis represents the average percent CPU utilization for TCP and UDP flows when a switch is placed in server-client path. As observed in figavgcpuclientnoswitch, the average CPU utilization remains unaffected by varying offloading parameters for UDP flows. However, for TCP flows, a significantly uniform and small percent of CPU utilization is observed.

In figavgcpuservernoswitch, the x-axis represents the offloading parameters, while y-axis represents the average percent cpu utilization.The fairly even distribution of cpu percent utilization with or without switch suggests that packet segmentation does not occur at the server end.

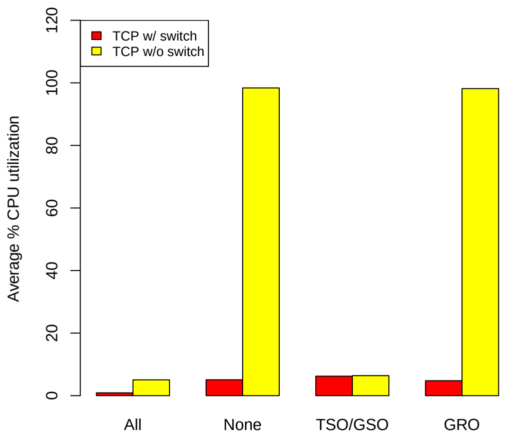

Figure 1: The x−axis represents offloading parameters while y−axis looks at average percent CPU utilization