Report: communcation jitter analysis, periodic peaks

Davide Rovelli, Michele Dalle Rive

04/2024

Objective

We analyse the latency disitribution in the communication between 2 bare-metal nodes in a datacenter. The objective of our experiments is to make packet processing latency of Linux-based hosts with modern NICs as *deterministic* as possible. This consists in reducing communication jitter to a minimum, where $jitter = \max(latency) - \min(latency)$.

Contrarily to usual low tail-latency experiments/systems which aim to reduce the common-case latency up to some percentile (e.g.$99.5\%, 99.9\%$) we want to minimise the 100% latency, even at the cost of (non-substantially) increasing the common-case.

Methodology

Benchmark

We run ping-pong tests across 2 hosts **A** and **B** by sending packets at a fixed sending rate. Each packet carries a packet ID and 4 timestamps:

- ts_1: timestamp just before sending the PING message in host **A**
- ts_2: timestamp as soon as the process in host **B** receives the PING packet.
- ts_3: timestamp just before sending the PONG message in host **B**
- ts_4: timestamp as soon as the process in host **A** receives the PONG packet.

We then calculate the difference between timestamps of successive packets to get the variations in latency. For example, $ts_1(33) - ts_1(32)$ where 33 and 32 are the packet IDs, represents the time interval between the PING send timestamp of packet 32 and the PING send timestamp of packet 33. If there's no jitter, it should be equal to the sending rate. The difference between $ts_1(33) - ts_1(32)$ and the sending rate represents the jitter for that pair.

Hardware setup

We use 3 different bare-metal clusters with the following configurations:

Cluster A 2x CloudLab xl170 nodes

- CPU: Intel Xeon E5-2640 v4 at 2.40GHz, 10 cores, 64GB RAM
- OS/kernel: Ubuntu 22.04.4 LTS / 6.6.19-060619-generic x86_64
- NIC/driver: Mellanox Connect-X 4 / mlx5

Cluster B 2x internal cluster nodes:

- CPU: Intel Xeon E5-2680 v4 at 2.40GHz , 28 cores, 64GB RAM
- OS/kernel: Ubuntu 22.04.4 LTS / 6.6.19-060619-generic x86_64
- NIC/driver: Mellanox Connect-X 4 / mlx5

Nodes are connected over ethernet, via a TOR switch with zero or minimal network load in order to only observe the endhost processing overhead.

Software setup

We use different *kernel bypass* methods in order to minimize the well-known overhead introduced by the classical network stack in Linux machines. We test the following:

- RoCE, specifically double-sided SEND/RECV RDMA with the Unreliable Datagram (UD) transport type
- eBPF XDP in two different modalities:
  - XDP-poll: packet is stored in a BPF map and polled by the application thread
  - AF_XDP: XDP socket type which handles TX and RX buffer management to the application
- Standard network stack over UDP

Packet size: *1024 bytes*

Analysis of periodic outliers

Here we show some of the results which include interestingly periodic latency outliers, which we refer to as "peaks". The objective of this analysis is finding the root causes of such spikes and possibly eliminating them.

Arrow-like peaks: interrupt coalescence

Long time-bound peaks