DNS Server Performance Tuning On Linux

DNS-OARC 42
8-9 February 2024
Charlotte, NC, USA

Jan Včelák
Senior Software Engineer at NS1
jan.vcelak@ibm.com

IBM

Context and goals

Context

NS1 is a managed DNS provider with anycast network.

We need to understand our server's performance.

General capacity planning.

Informed decisions when refreshing hardware.

Benchmark

Authoritative service only.

QPS as a performance metric.

Performance estimate is sufficient for this purpose.

Optimizations

Where should we focus our effort?

Environment configs, tunables, and toggles. Operating system, NIC, support services, etc.

Software (application).

Design of the benchmark

Technique

Run DNS server in echo mode.

Run the benchmark.

Get confidence in the setup. Identify misconfiguration. Establish performance baseline.

Run DNS server in production mode. Identify software problems.

Software

Custom DNS server.

kxdpgun to generate traffic.

System services (disable irqbalance, beware of conntrack, beware of CPU frequency scaling, etc.).

Tuning and monitoring tools (ethtool, taskset, set_irq_affinity.sh, ss, top, etc.).

Hardware

Two servers connected through switch.

- DNS server:
  AMD EPYC 7302P (16 cores, 2 threads per core)
  Mellanox Technologies MT27800
- Traffic generator:
  Intel Xeon Gold 6326 CPU (2 sockets, 16 cores per socket, 2 threads per core)
  Intel Corporation Ethernet Controller E810

DNS is dominated by small packets at high rates

DNS prefers UDP.

Processing each packet adds a lot of overhead. It is difficult to utilize max PTU.

Let's assume we have a 10 Gbit network adapter.

DNS, IPv4, UDP, and average DNS message size of 100 bytes.

Each packet will use 166 bytes of the bandwidth.

If we manage to use the full line rate, we will have to process **7,530,120 queries per second**.

With protocols that can use large packets (e.g. HTTPS), the packets can be up to 1500 bytes (without Jumbo frames) and each packet will use 1538 bytes of the bandwidth.

Max rate will be just **812,744 packets per second**.

Packet receive and transmit path

NIC receives a packet, adds the packet into the RX queue, and generates hardware interrupt.

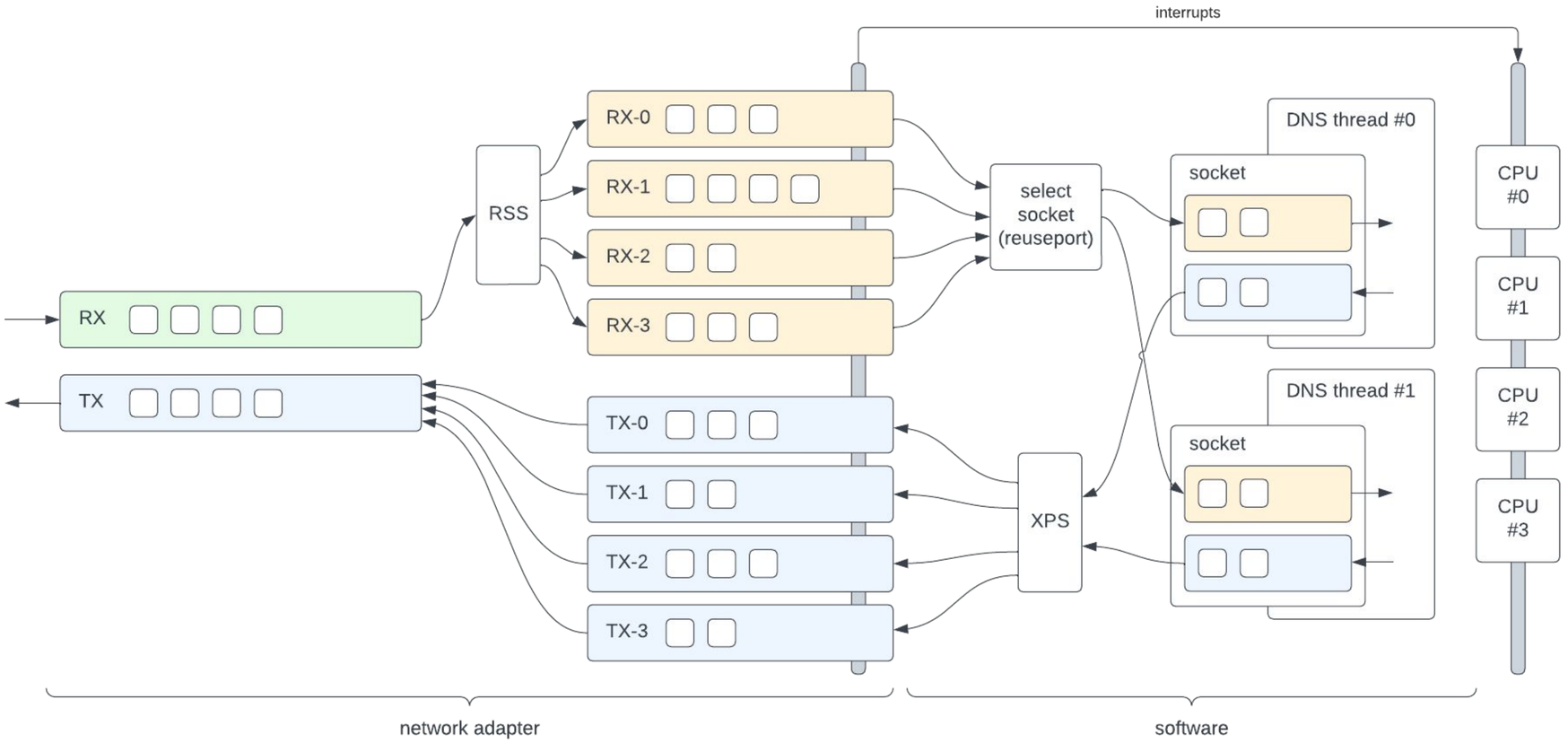
OS handles the interrupt, schedules data transfer, copies the packet into memory, determines destination socket, moves packet into application's socket buffer.

Application reads the query and sends the response from/into the socket buffer.

OS determines the TX queue, transfers the data to the NIC.

NIC sends the packet, generates interrupt.

OS marks the transfer as complete and schedules a next one.



Environment optimizations

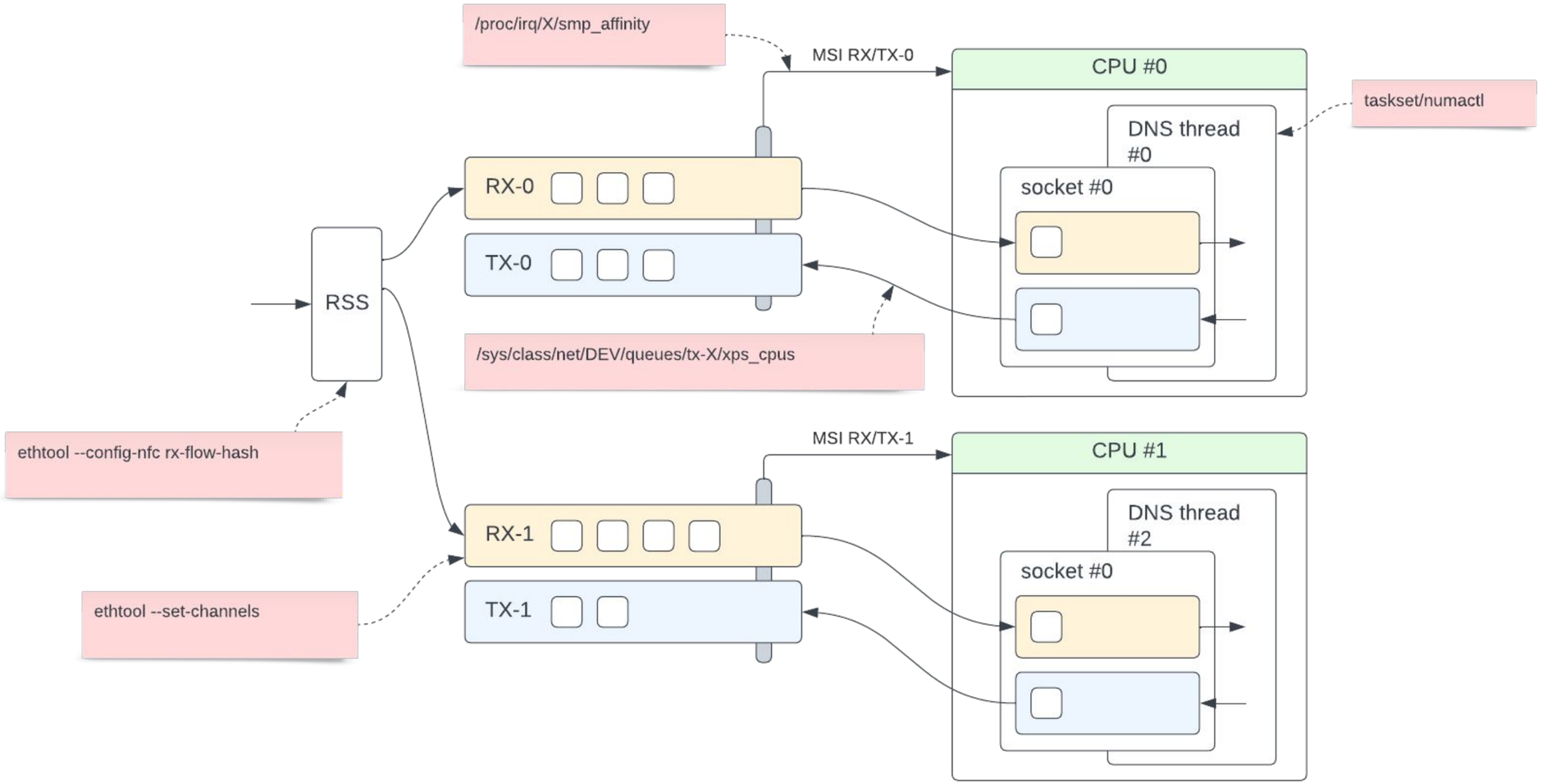
Goals:
- Even distribution of the load.
- Avoid expensive context switches.

Network parameters:
- Configure Receive-Side Scaling (RSS)..
- Number of RX/TQ queues.
- RX/TQ queue sizes.

Application:
- Pinning application threads to CPUs.
- Set interrupt affinity for the RX/TX queues.
- Enable Transmit Packet Steering (XPS).

Observations:
- One NIC per CPU seems ideal to avoid contention. If needed, adjust the size of the buffers.
- Other NIC tunables (interrupt coalescing, hardware offloads, etc.) seems to have diminishing effect.

Utilities:
- ethtool
- set_irq_affinity.sh (Intel)
- taskset/numactl



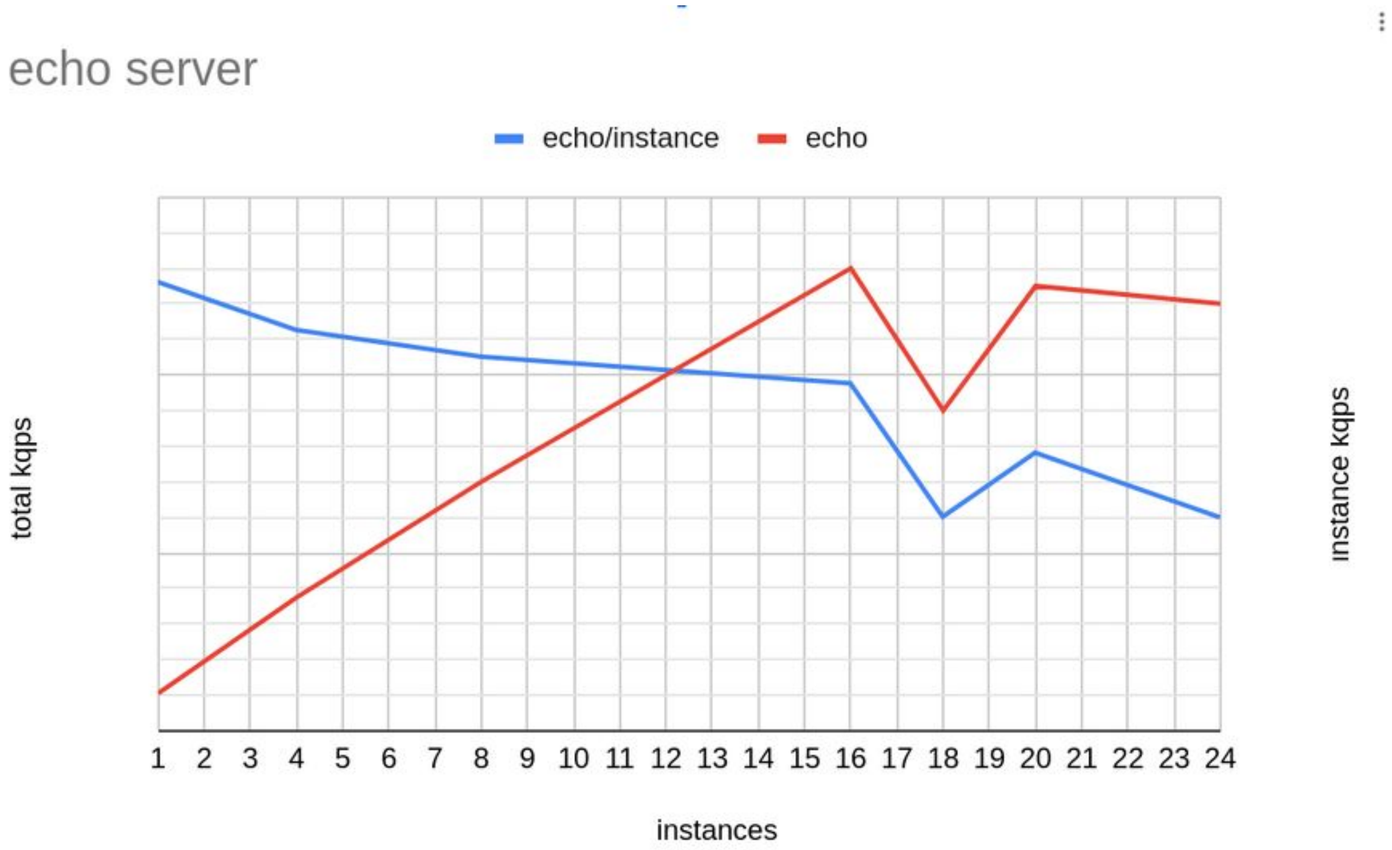
How many server threads to run?

Not all CPUs are the same.

In the example, the performance increases up to the number of physical CPUs and then drops.

Adding more cores adds significant contention.

Note that the performance per core is not constant either.



Metrics and indicators of contention

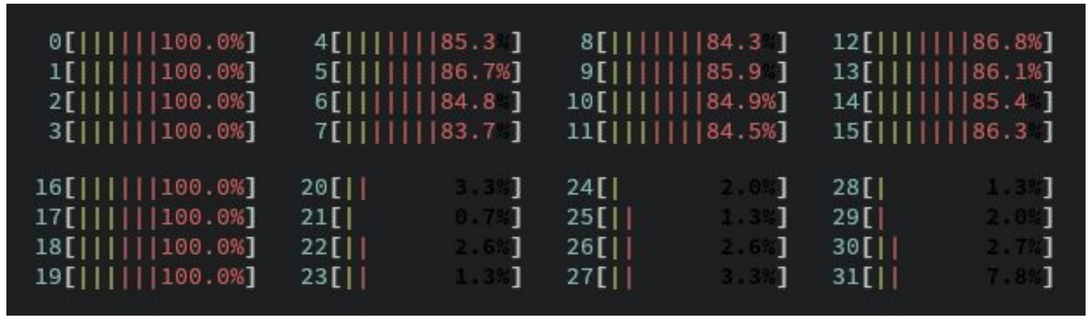
Useful metrics

- CPU load (top)
- Interrupt counters (/proc/interrupts)
- NIC packet drops (ip -stats link)
- socket buffer drops (ss --memory)

Example of CPU contention

16 CPUs, 32 cores (HT)
20 DNS server instances
20 TX/RX queues

Cores sharing the physical processor (0-4 and 16-19) are maxed out and packets are being dropped.



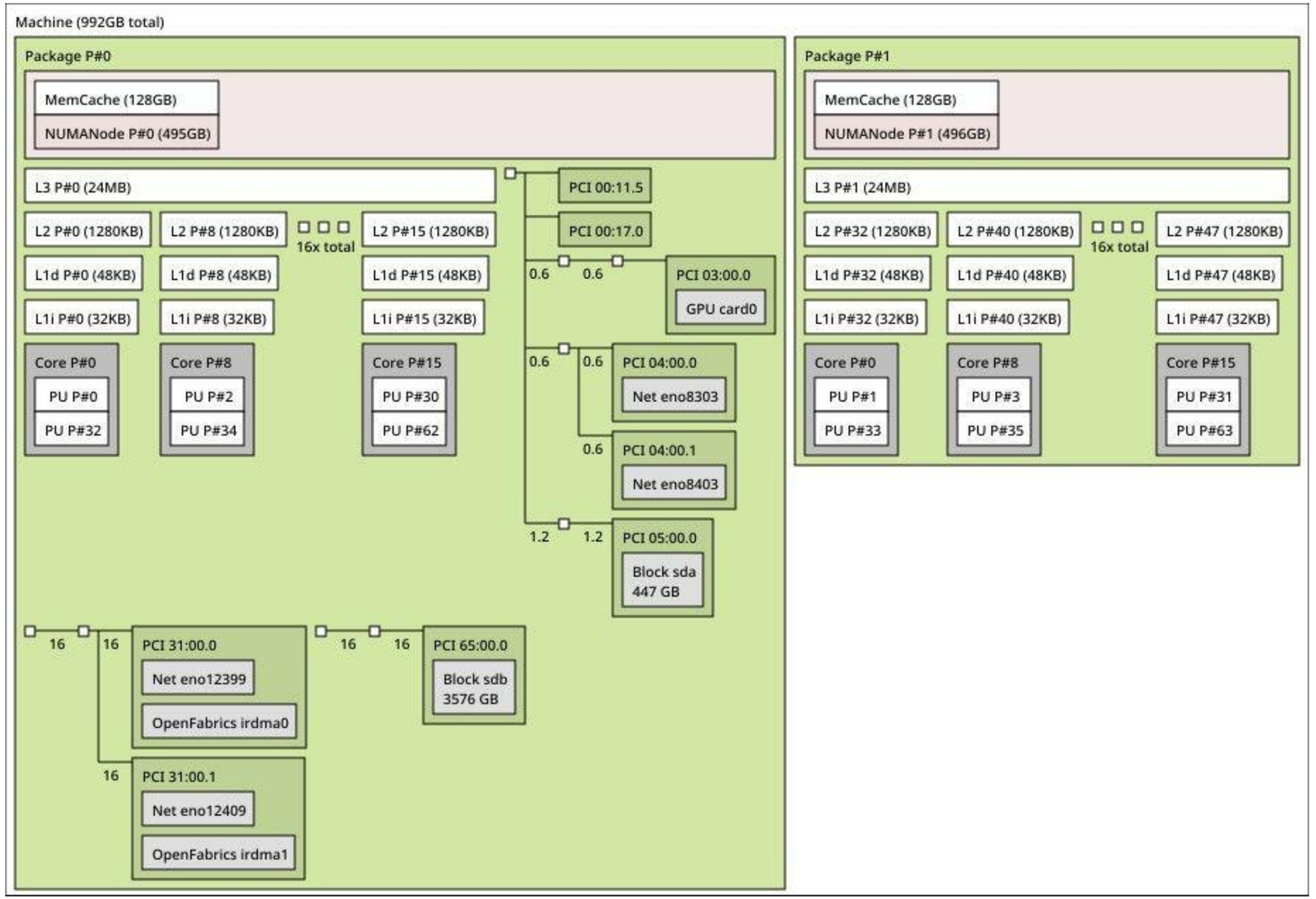
Beware of NUMA

The DNS server most likely needs to be adapted to work well on Non-uniform Memory Access (NUMA).

NUMA makes memory access time dependent on the distance from the CPU.

DMA transfers from the NIC are affected. There will be a separate PCI controller for each node.

Tools: lscpu, lstopo



Learned Lessons

Environment configuration has serious impact:
- Tunable: CPU affinity, RX/TX queue configuration, buffer sizes, etc.
- Unavoidable: Firewall, endpoint security, etc.

Measuring the performance should be a continuous effort:
- The environment is changing constantly.
- Long time tracking is valuable.
- Great value for optimization work.

Capacity planning:
- Do not oversimplify (e.g. performance per core × core count = total capacity).
- Know your hardware, beware of SMT (e.g. Hyper-Threading), beware of NUMA.
- Be realistic.

Our own learnings:
- We run too many DNS server threads.
- The numbers for capacity planning are not up-to-date.
- There is low hanging fruit for optimizations (environment and software).
- We should be more systematic in the measurements.
- Performance measurements is fun and a great learning experience.

Resources

Scaling in the Linux Networking Stack
https://docs.kernel.org/networking/scaling.html

set_irq_affinity.sh (within Intel Ethernet Drivers and Utilities Files)
https://sourceforge.net/projects/e1000/files/ice%20stable/1.12.7/

NUMA Memory Policy – The Linux Kernel Documentation
https://docs.kernel.org/admin-guide/mm/numa_memory_policy.html

Thank you