Routing for AI training clusters

Survey of the problem space

Petr Lapukhov, NVIDIA

Agenda

- Historic detour
- The dawn of GPU networking
- Summa topologia
- On 100K GPU and beyond

Historic Detour

The glorious days of 2012

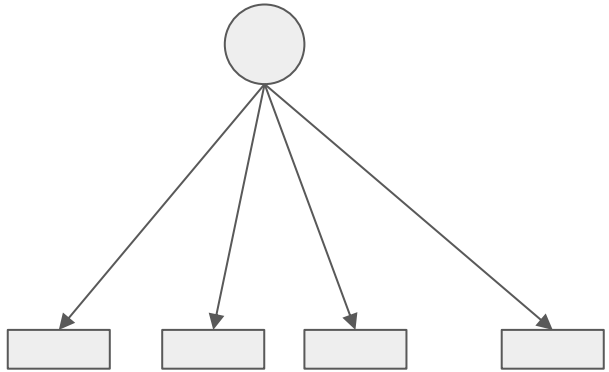
We had not AI craze back then (only SDN)

- …Yet we already had 100K-port clusters!
- Except those were CPUs :)

Typical layout:

- Online-services: **query-response** traffic
- **Data-mining** (map-reduce/hadoop)
- …

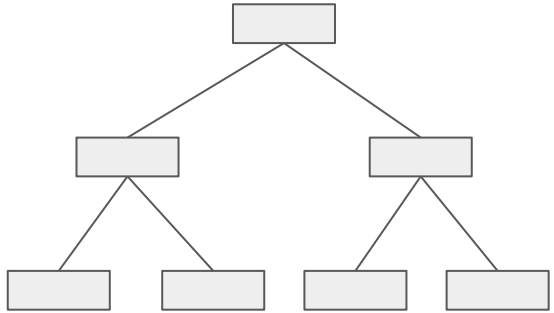
100-200W / server -> 10-20 MW for 100K servers



The victory of L3

- L2 had serious resiliency and scaling problems
- “Routable L2” (TRILL, FabricPath) did not gain momentum
- **Routed** tree architectures (IGP and/or BGP) won
- Flow-based hashing (ECMP) with all its joys

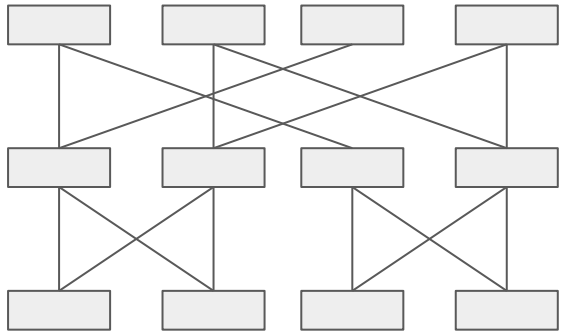
Surprisingly, (!) BGP became a norm in data-center



E/W traffic and fat-trees

- Map-reduce data-shuffling → all-to-all traffic (mappers)
- Bandwidth: **fat tree with many paths**
- Switch radix: 64-128 leading to **3 or 5 tier trees**
- Typical fan-out: 4-way or 8-way

Importantly: still **one big (fat) tree** for all traffic (online/data-mining)



A word on the transport

- TCP was undisputed - with various tunings (ECN, DC-TCP…)
- NIC-assisted offloads (GSO, LRO, checksum)
- Memory bandwidth (copies) + CPU cores
- The incast (fan-in) and speed mismatch problems

Despite all TCP shortcomings, RDMA wasn't popular due to… many reasons. TCP just mostly worked and people get used to it.

RoCE v1 was around, mostly used for storage. RoCE v2 (RRoCE) started taking shape.

The dawn of GPU networking

From TCP to RDMA

- GPU-based systems started entering scene starting 2014-2015
- Moving data to GPU was via CPU + NIC + host memory
- Then came GPU direct + RDMA NIC (CPU orchestrated)

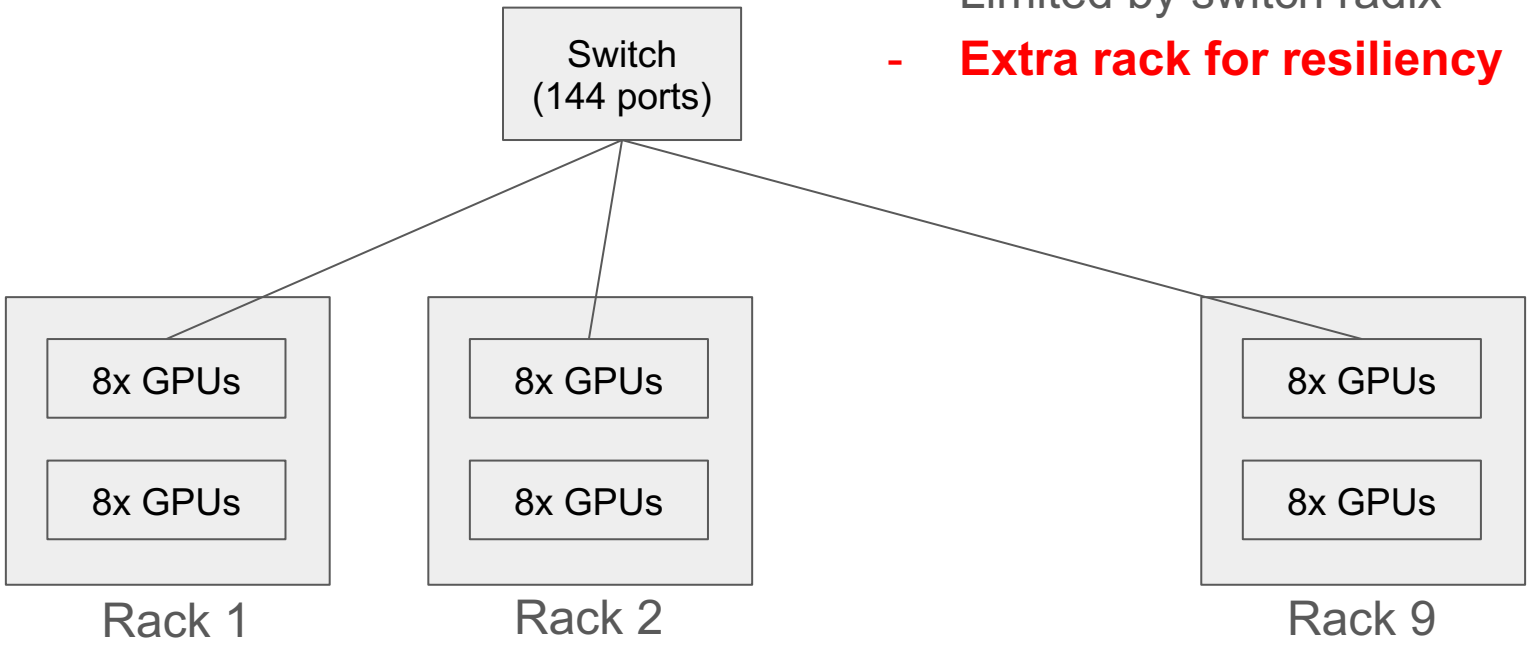
Tension points

- NIC-based transport - hard(er) to inspect and debug
- The RoCE vs IB debate (same RDMA behind)
- **Big fears of RoCE congestion spreading in multi-hop networks**

Your very first E/W cluster™

144 GPUs in one “pod”

- One-hop network to “deal” with RoCE flow control
- Does not need to be routed
- Limited by switch radix
- **Extra rack for resiliency**



The collective communications (NCCL et al)

Train on many GPUs in parallel, because you need the FLOPs

- Training **parallelism** comes from either “sharding” or “replicating”
- In either case we get ‘gang’ of GPUs communicating symmetrically

We encounter familiar “logical” and “transport” comms patterns

- **Logical:** All-to-all/all-reduce/all-gather/reduce-scatter
- **Transport:** Ring, halving-doubling pattern, binary trees

The collective communicating replace Hadoop’s map-reduce flow graphs

The great fabric schism

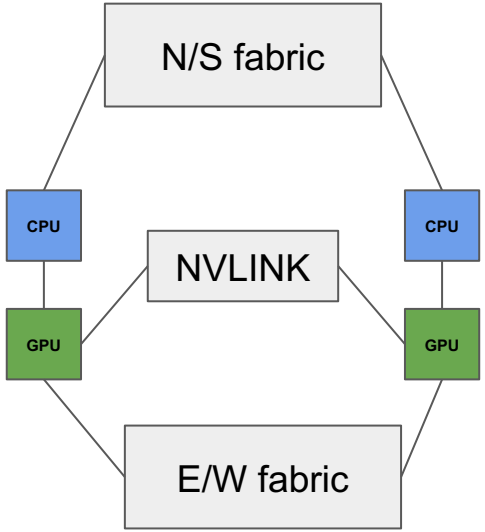
- So now we have a CPU + NIC and a GPU + NIC
- Can they use one NIC?
- Can they plug into same fabric?

Long-story short, many decided to **split the fabrics**

[1] NVLink private (always private - for now)

[2] GPU scale-out, or the E/W fabric (RDMA traffic)

[3] CPU N/S - storage and management



Summa topologia

The new world order and its problems (1)

The flow load-balancing problem:

- RDMA NICs push at line rate of 100G/200G/400G NICs
- **The real “elephant” flows - do not play well with ECMP**
- The first line of defence is “connection split” - add more flows to the network. But that only improves as sqrt(N)

The new world order and its problems (2)

Effects from the “collective” comms:

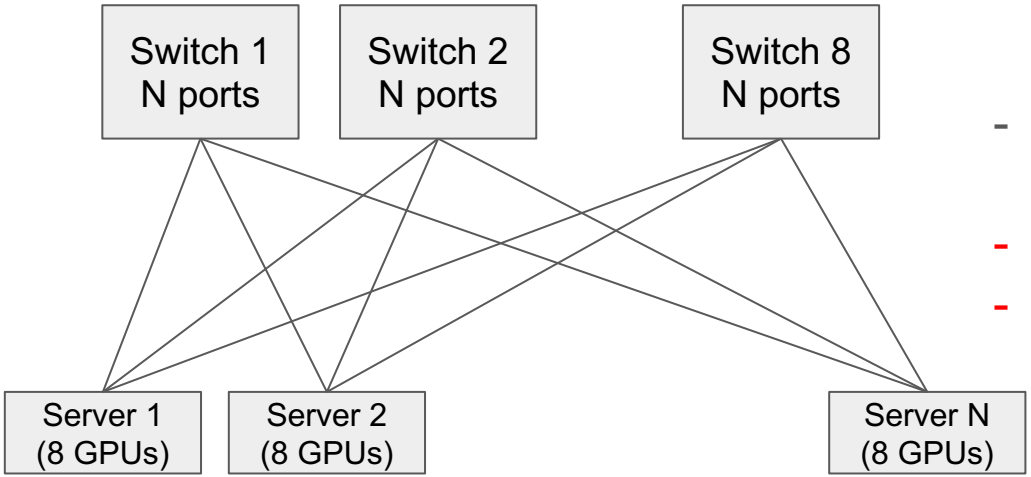
- **Latency** accumulation in “ring-based” collectives
- **Congestion** in “log-” collectives (trees, halving-doubling)

The new world order and its problems (3)

Resiliency:

- Effect of failures much more pronounced - “collective” fails together
- Capacity losses more pronounced with elephant flows - **imbalances**

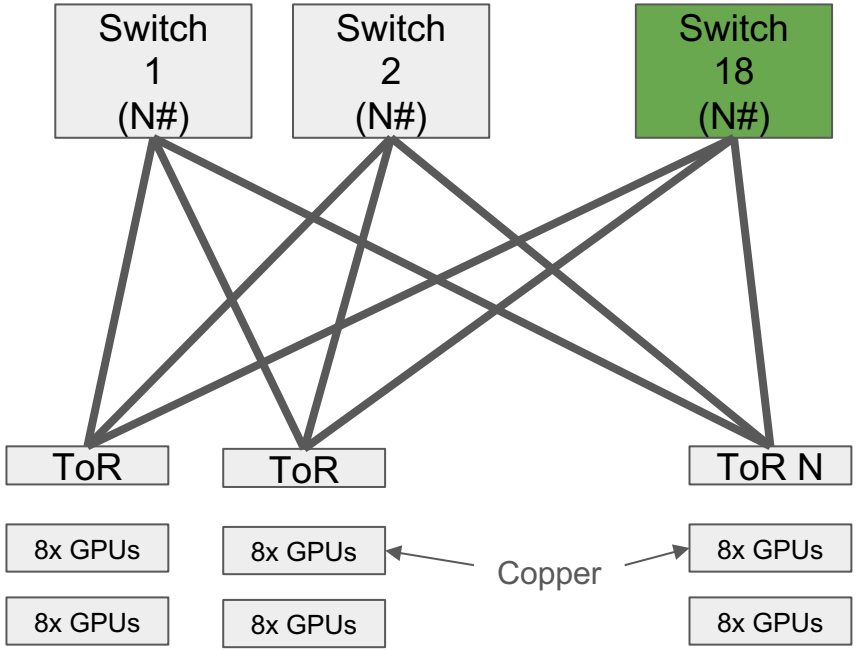
First load-balancing idea: rails



- Still one-hop for RDMA traffic
- Load-balancing by the collective library: NCCL **places flows on different rails**
- **Rank-disjoint planes** - all-to-all traffic has to cross over NVLink
- **Optics in the NICs!**
- **No resiliency to switch failures**

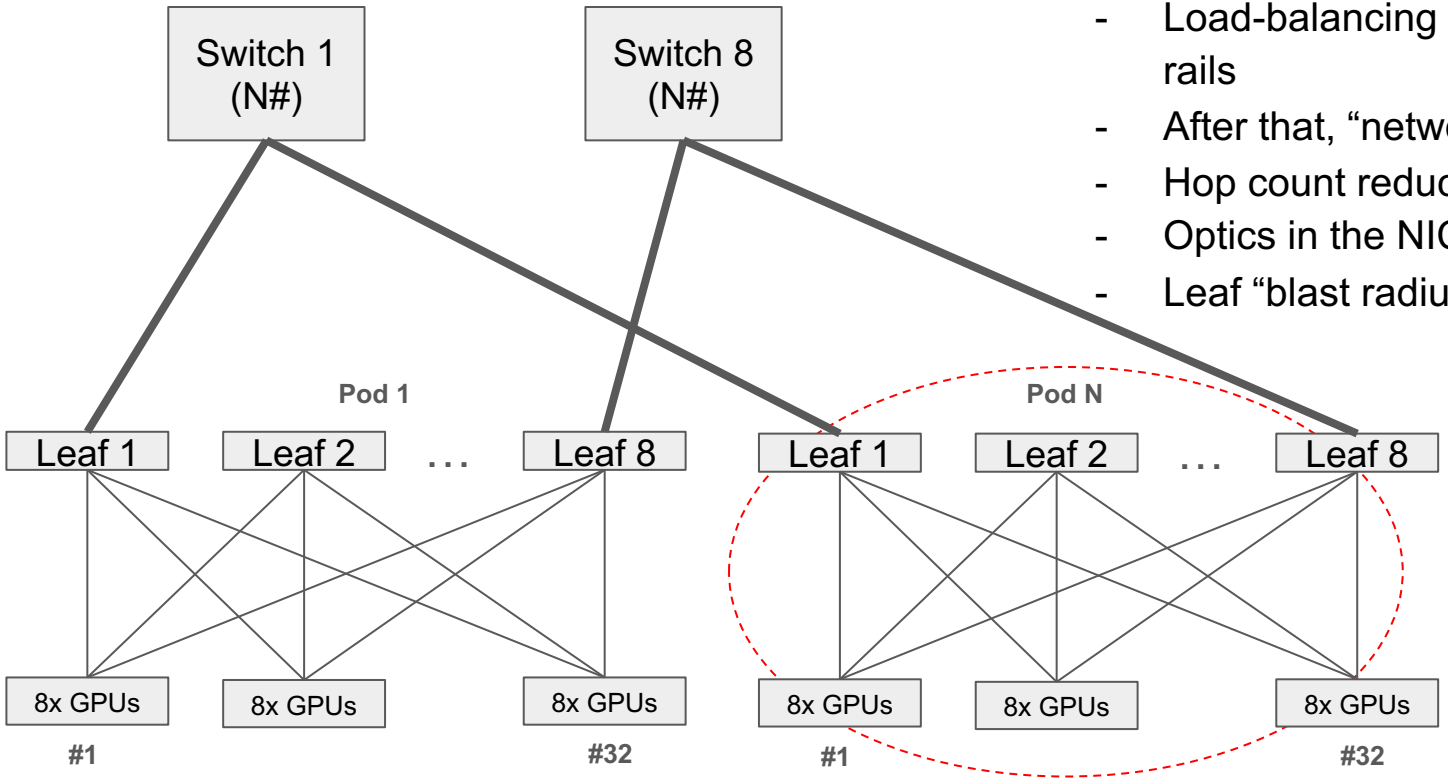
Here we get Nx8 GPUs

The fat tree for E/W, redux: ToR-based design



- Requires routing + RoCEv2
- ToR does **flow load-balancing** (ECMP or something better)
- Uplinks may be 2x or more faster than downlinks - better stat-muxing
- NIC connections are **copper**
- There is more uplinks per ToR then downlinks - **resiliency**

The fat tree for E/W, redux: Rail optimized design



- Load-balancing from **servers** using rails
- After that, “network routing”
- Hop count reduces inside “pods”
- Optics in the NIC, end-of-row leafs
- Leaf “blast radius”

Fine-grained load-balancing & adaptive routing

By now we see that half of the problems is load-balancing :)

- Can we **split** elephant flows into smaller units?
- Flowlets, packet-spraying, etc
- Adaptive routing: distribute load based on network utilization
- Supporting out-of-order packets in the endpoint

[1] In-network vs. in-NIC

[2] Oblivious vs. adaptive

[3] No standards, really

On 100K and beyond

Back to 100K clusters, but now with GPUs

Utility power becomes a precious resource

- Was ~100-200W per single-CPU, now ~1KW per GPU
- 100-150 MW for a 100K cluster!

Reliability now even more painful

- One big training job
- Resiliency can be built in training but there are limits

The network power wall

Network power starts to matter

- It’s not 10G links anymore…
- Few KW per switch
- ~ 22-25W for 1.6T OSFP

You end up with 10-20 of MW for the network for 100K+ cluster, climbing into 15% zone

The large-scale “yet-power-efficient” network

Reducing number of fat-tree tiers

- Fat-tree scale is O(N * log(N))
- Want as shallow of a tree as possible
- E.g. 2 tier fat-tree - yet covering the 100K scale

Shallow fat-tree:

- Requires running switches at maximal “fan-out” - e.g. 512x ports with 51T switch*
- Now we have lots of thin links (100G?!), how we handle elephant flows?!

Requires fine-grained load-balancing from the host (NIC)