Design Challenges for Combining Compute and Networking

Dave Oran
Network Systems Research & Design

NETWORK SYSTEMS RESEARCH & DESIGN

Npvember 9, 2018
Computing in the Network Side Meeting – IETF103

Warning

I am way better at questions than I am at answers!

We have a very large design space

- Incremental or clean-slate or something in between
- Do we need to limit the programming model, or just loosely provide some APIs?
- How much can we leverage the Cloud tooling that already has a large developer community?
  - If we don't, we need a much longer gestation period, but possibly a much bigger win due to reduced complexity
- Need to address the barriers that mis-align the interests of people tasked to develop/deploy applications, and people tasked to instantiate and manage the network.
- What if anything is "different" about "network functions"?
  - If the answer is "nothing" then can we treat most network functions as just a compute graph (including routing)

Exploiting Heterogeneous Compute

- Rich zoo of (mostly) incompatible platforms:
  - Smart NICs with embedded FPGAs
  - Switch ASICs
  - GPUs and FPGAs on Server Hardware
  - Switches with FPGA co-processors
- How many different programming models do we have to deal with?
  - General purpose procedural code
  - Non-Turing Complete languages like P4
  - Functional Programming – Erlang, Haskell, etc
  - Interpreted languages – Python, Java etc.
- Some pretty sticky problems in
  - Virtualization and sharing of the hardware resources
  - Currently there's a large penalty for not running at wire rate when implemented in switches

Alternative models for distributed programming

- Regular RPC or Restful Web transactions targeted directly at the network devices
- Packet interception schemes like NFV
- Pre-computed function chains like SFC

Or something more radical?

- Resurrect active networking - programs in the packet headers
- Resurrect Dataflow computation on the network now that the network is really fast – turns asynchrony from a problem to an advantage

Speculations #1- Datacenter and Edge

- Can we do joint optimization of routing / traffic engineering / workload placement?
  - Today datacenters place load based on a simple/naïve model of topology (same rack versus different rack) and costly instance instantiation
  - In COIN environment, can we assume instantiation is cheap and just do an agile global optimization?
- Can we do this cheaply and highly distributed?
  - Or are we restricted to mostly-homogeneous cases – all on the edge or all in the data center back end?

Speculations #2 – Edge-specific issues

- We think we need distributed trust, but what does this really entail?
  - Can we protect long-lived keys acceptably in these edge compute resources? Maybe we need some sort of explicit delegation from the cloud so edge resources only need ephemeral keys
  - Is the trust-schema approach in being explored in the ICN world workable here? Seems like it should be but lots to still validate by actually building stuff.
- Are the very-low end devices conntected to the edge "first class" clients in COIN, or do we need some separate networking support for them?
  - Some others are placing agents on the edge boxes to proxy for them
  - This is somewhat at odds with the "everything is a peer node" approach. What do we think?

Final thoughts

- Where are the "big win" applications of COIN at the edge?
  - Privacy-preserving analytics could help to justify placing non-trivial amounts of computing at the edge
  - Lots of enthusiasm for automotive applications, but unclear still (to me) how the necessary physical infrastructure gets deployed
  - How to avoid falling into the "just the latest CDN re-invention" trap?
- Where are they in the data center?
  - Map-reduce or ML training optimizers?
  - Really fast consensus protocols?
  - Scalable KV Stores with caches in the network devices?
- Are DC and edge enough similar or do we really have two independent problem spaces that don't need or want a common solution set?