Networking Recap
Storage Intro

CSE-291 (Cloud Computing), Fall 2016

Gregory Kesden

Networking Recap
Storage Intro

Long Haul/Global Networking

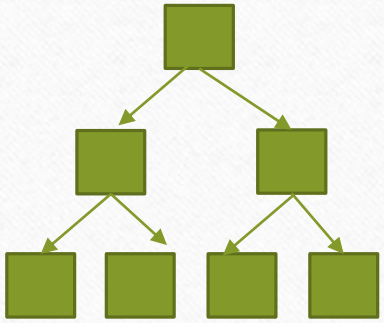
- Speed of light is limiting; Latency has a lower bound (.)
- Throughput is a cost, but not otherwise a limitation
  - Running more fiber optic is technologically reasonable, within reason
  - Demand will drive growth

Vs. Networking Within A Data Center

- Speed of light not limiting, short distance
- Serialization can be limiting, as bits clocked into and off of media
- Parallelization helps, but quickly limited in practice
- Bandwidth (Throughput) needs to be managed, both by distributing load and relieving bottlenecks
- Task drives access pattern, drives hot spots

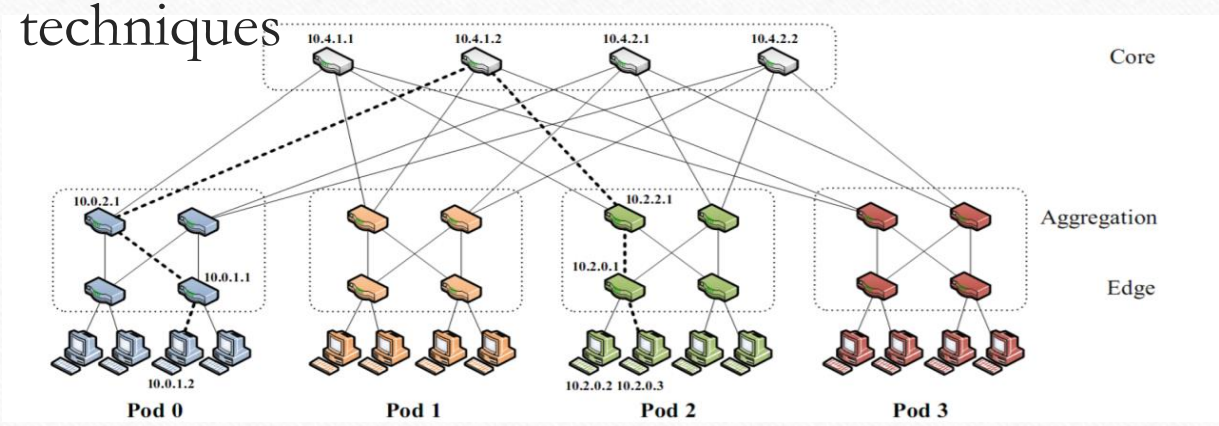
Traditional 3-Tier Network

- Convenient for distribution
- All links same width
  - More links at lower layers = More bandwidth lower layers
- Works if strong locality at leaf (access layer)
- Challenged if leaf-to-leaf up-and-down patterns
- Challenged if request distributed and wait reply (inflow)
  - Maximizing work within time budget means replies at same time collide



Fat Tree Topology

- Multi-Rooted Tree = All paths same width
- Only useful if load distributed
  - Doesn't work if all pick same spanning tree
- Can distribute via Layer-1,-2, or -3 techniques



Utilizing Multiple Paths

- Layer-2
  - Forwarding tables; Consider PortLand
  - Hard to be adaptive
- Layer-3
  - ECMP; Can do via routing
  - Suffer collision of paths under load
  - Slow to adapt, slow implementation
  - Handles failure slowly
- Layer-4
  - Schedule flows
    - Not all flows are equal
    - Can change
    - Big vs small can't be balance
  - DCTCP; Multiplex flows
    - Positive Feedback to balance, not based upon interpreting drops.
    - Adaptive, balanceable

PortLand

- Uses FatTree Topology
- Provides way of using multiple paths
- Separates Identifier (IP address) from Locator (PMAC)
- Facilitates migration (Benefit over other solutions)
- Fabric Manager enables failure management, mapping, etc
- Link Discovery enables Auto-Configuration

Clos Networks

- Obvious benefit to flatter, all-to-all Connectivity
- Presently enables 2-Layer topologies in many cases
  - Fabric has limits. Only possible to a point.
  - Scale may more commonly outgrow fabric in near future leading to migration b ack to 3-Tier
- Manage scale limitations by using locally within pods, among cores, etc.
  - Provides local density where needed, without blowing up interconnectivity scale globally

Software Defined Networks (SDNs)

- Separate Control Plane (Management) from Data Plane (Forwarding)
- Enable central management of entire control plane
  - Essentially makes a programmable network
- Enables management applications with appropriate abstraction for use case (enterprise, public cloud, etc) and management goal (traffic management, isolation, etc)
- Dramatically reduces management overhead, correctness, and security
  - This is a "killer app", by itself, at scale
- Critically important for virtualization
  - Enable on-demand management, and real-time adaptation to failure.
- Replace multiple types of devices with single, generic, programmable devices

Networking Recap
Storage Intro

Types of Storage

- General Purpose File System
- Special Purpose File System
- NoSQL Database (Key-Value Store, Column-oriented, Etc)
- Relational Databases, i.e. SQL
- In Memory (Critical)

General Purpose File System

- Emulate traditional file systems at larger scale and/or distributing near users
  - E.g., Manage named user data with hierarchical access and protections,
- Andrew File System (AFS) is classical global example, many more like NFS approximate at DC or campus scale
- Friendly for end users, but how useful?
  - Commonness of global use case
  - Can be very distant with reasonable latency.
    - Disk = 10mS; Network = 10mS/1,000mi (very approximate)
- High cost
  - Synchronization, concurrency, etc

Special Purpose File System

- Relax constraints to limit complexity
  - Accessing programmatically? Use index, drop directory hierarchy.
  - Upload-based creation? Version and don't support edits. Limits concurrency problem to version number
  - Logging? Append-only writes. Limits concurrency problem to allocation.
  - Single application? Access model limited to roles, let app manage permissions
  - Stale okay? Easier replication.
  - Etc.
- Relieving concurrency problems
  - Improves caching, replication
  - Decreases latency

NoSQL Databases

- Break up limitations associated with table (relation)
  - CAP: Trade consistency for accessibility and partition tolerance.
  - Speed at scale
- Not likely ACID
- Can do some SQL-like things, but some are hard, e.g. generalized join
- Many kinds, e.g. key-value, column-oriented, etc

In-Memory

- When reacting to human users in real time, time budget is limited
  - Trip from client to service, trip from front-end to back-ends, processing, trip from back-ends to front-end, processing, trip back to client, etc.
- There isn't time to constantly hit disk in many ways
  - Would need a huge amount of disk time to be able to sustain throughout at scale
  - What is reasonable response time for human user? 0.5 second (500 mS)?
    - 75mS round trip to east coast
    - The rest is data center latency, queuing for services, and actual work
- Best solution is to have the answer waiting
  - Or, at least the components of the answer
  - Sadly, sometimes block for slowest component