CS434/534: Topics in Network Systems

Cloud Data Centers: Topology, Control; VL2

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/

Acknowledgement: slides contain content from conference presentations by authors of VL2.
Outline

- Admin and recap
- Controller software framework (network OS) supporting programmable networks
  - architecture
  - data model and operations: OpenDaylight as an example
  - distributed data store
    - overview
    - basic Paxos
    - multi-Paxos
    - raft
  - south-bound: consistent network updates
- Cloud data centers (CDC)
Admin

- PS1 posted on the Schedule page
- Please start to talk to me on potential projects
Recap: Raft

- Leader-less => leader based

- Basic leader election mechanisms
  - term, heartbeat, finite-state machine, receives vote from majority

- Basic commitment of a log entry
  - receives confirmation from majority
Recap: Raft Safety

- **Raft safety property:**
  - If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders
  - No special steps by a new leader to revise leader log

- **Solution**
  - Use a combination of election rules and commitment rules to achieve safety
Recap: Big Picture

Key goal: provide applications with high-level views and make the views highly available (e.g., 99.99%), scalable.

Key component - data store:
- Data model
- Data operation model
- Data store availability

Program

logically centralized data store

Network View
Service/Policy
NE Datapath
NE Datapath
Discussion

- What should happen in a Net OS when a link weight is changed
Recap: Big Picture

Assert: TcpDsd==22

true

false

Read: EthSrc

Read: EthDst

Read: EthDst

(pathTable, 1)

(pathTable, 2)

(pathTable, 3)

(pathTable, 4)

(pathology())

(pathology())

(pathology())

(pathology())

Dependency table (aka inverted index)

<table>
<thead>
<tr>
<th>Component</th>
<th>Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>(hostTable, 1)</td>
<td>[false, 1]</td>
</tr>
<tr>
<td>(hostTable, 2)</td>
<td>[false, 1, 2]</td>
</tr>
<tr>
<td>(hostTable, 3)</td>
<td>[false, 3]</td>
</tr>
<tr>
<td>(hostTable, 4)</td>
<td>[false, 3, 4]</td>
</tr>
<tr>
<td>topology</td>
<td>[false, 1, 2], [false, 3, 4]</td>
</tr>
</tbody>
</table>
Recap: Example Transaction

- Link weight change => path for flow a→d change:
  - from a → b → c → d
  - to a → e → f → d

- A high-level transaction can generate a set of operations (ops) at local devices.
- The ops should be executed with some order constraint (dependency graph)
Recap: Example Transaction: Updating a Set of Flows

Assume each link has capacity 10

Fi: b: means that flow i needs b amount of bw

A transaction can be more complex, and hence coordination can be more complex as well.

Example from Dionysus
Dependency Graph w/ Resources

(a) Current State

(b) Target State

Mv.
F3
10

Mv.
F3
10

S2-S5: 0

Mv.
F4
5

S1-S5: 0

Mv.
F2
5

Mv.
F1
5

S1-S5: 0

Mv.
F1
5
Dependency Graph Scheduling

operation

resource

before

release N

demand N
Potential Project: Continuous, Consistent Network Updates

- Discussion: how to define the problem, what is a good data structure, ...

![Diagram of network updates]

(a) Current State

(b) Target State
Outline

- Admin and recap
- Controller software framework (network OS) supporting programmable networks
- Cloud data center (CDC) networks
  - Background, high-level goal
The Importance of Data Centers

- Internal users
  - Line-of-Business apps

- External users
  - Web portals
  - Web services
  - Multimedia applications
  - Cloud services (e.g., Azure, AWS, ...)
Datacenter Traffic Growth

Today: Petabits/s in one DC
- More than core of the Internet!

### Data Center Costs

<table>
<thead>
<tr>
<th>Amortized Cost*</th>
<th>Component</th>
<th>Sub-Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>~45%</td>
<td>Servers</td>
<td>CPU, memory, disk</td>
</tr>
<tr>
<td>~25%</td>
<td>Power infrastructure</td>
<td>UPS, cooling, power distribution</td>
</tr>
<tr>
<td>~15%</td>
<td>Power draw</td>
<td>Electrical utility costs</td>
</tr>
<tr>
<td>~15%</td>
<td>Network</td>
<td>Switches, links, transit</td>
</tr>
</tbody>
</table>

---

*3 yr amortization for servers, 15 yr for infrastructure; 5% cost of money

Server Costs

Ugly secret: 30% utilization considered “good” in data centers

- Uneven application fit
  - Each server has CPU, memory, disk: most applications exhaust one resource, stranding the others

- Long provisioning timescales
  - New servers purchased quarterly at best

- Uncertainty in demand
  - Demand for a new service can spike quickly

- Risk management
  - Not having spare servers to meet demand brings failure just when success is at hand

- Session state and storage constraints
  - If the world were stateless servers, life would be good
Goal: Agility - Any Service, Any Server

- Turn the servers into a single large fungible pool
  - Dynamically expand and contract service footprint as needed

- Benefits
  - Lower server component cost
  - Achieve high performance and reliability
  - Increase service developer productivity

Agility: The same of most infrastructure projects
Achieving Agility

- Workload management
  - Means for rapidly installing a service’s code on a server
  - Virtual machines, disk images, containers

- Storage Management
  - Means for a server to access persistent data easily
  - Distributed filesystems (e.g., HDFS, blob stores)

- Network
  - Means for communicating with other servers, regardless of where they are in the data center
Datacenter Network Ultimate Goal

Provide the illusion of "One Big Switch"

10,000s of ports

Compute

Storage (Disk, Flash, ...)

Datacenter Network Ultimate Goal

Provide the illusion of "One Big Switch"

10,000s of ports

Compute

Storage (Disk, Flash, ...)
Outline

- Admin and recap
- Controller software framework (network OS) supporting programmable networks
- Cloud data center (CDC) networks
  - Background, high-level goal
  - Traditional CDC vs the one-big switch abstraction
Conventional DC Architecture

Reference – “Data Center: Load balancing Data Center Services”, Cisco 2004
Conventional DC: Topology Problem

Heterogenous server-to-server capacity
• Fundamentally a tree, the higher up in the tree, the more potential competition on resources, limiting any server for any service
**Conventional DC: Topology Problem**

**Poor reliability**

- Fundamentally a tree, link failures in top of the tree can lead to large fraction loss/reliability issues
Partition by IP subnet limits agility
• For a VM to move to a different subnet (e.g., to use the resources), the VM’s IP address must change.
Discussion: L2 vs L3

Key
- **CR** = Core Router (L3)
- **AR** = Access Router (L3)
- **S** = Ethernet Switch (L2)
- **A** = Rack of app. servers

Reference – “Data Center: Load balancing Data Center Services”, Cisco 2004
Layer 2 vs. Layer 3

- **Ethernet switching (layer 2)**
  - ✓ Fixed IP addresses and auto-configuration (plug & play)
  - ✓ Seamless mobility, migration, and failover
  - ✗ Broadcast limits scale (ARP)
  - ✗ Spanning Tree Protocol

- **IP routing (layer 3)**
  - ✓ Scalability through hierarchical addressing
  - ✓ Multipath routing through equal-cost multipath
  - ✗ More complex configuration
  - ✗ Can’t migrate w/o changing IP address
## Layer 2 vs. Layer 3 for Data Centers

<table>
<thead>
<tr>
<th>Technique</th>
<th>Plug and play</th>
<th>Scalability</th>
<th>Small Switch State</th>
<th>Seamless VM Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layer 2:</strong></td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Flat MAC Addresses</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Layer 3:</strong></td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>IP Addresses</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Admin and recap
- Controller software framework (network OS) supporting programmable networks
- Cloud data center (CDC) networks
  - Background, high-level goal
  - Traditional CDC vs the one-big switch abstraction
  - VL2 design and implementation
Measurements Informing VL2 Design

- **Data-Center traffic analysis:**
  - Traffic volume between servers to entering/leaving data center is 4:1
  - Demand for bandwidth between servers growing faster
    - Network is the bottleneck of computation
  - Traffic patterns are **highly volatile**
    - A large number of distinctive patterns even in a day
  - Instability of traffic patterns
    - Cannot predict traffic easily

- **Failure characteristics:**
  - Pattern of networking equipment failures: 95% < 1min, 98% < 1hr, 99.6% < 1 day, 0.09% > 10 days

- **Flow distribution analysis:**
  - Majority of flows are small, biggest flow size is 100MB
  - The distribution of internal flows is simpler and more uniform
  - 50% times of 10 concurrent flows, 5% greater than 80 concurrent flows
Discussion

- How may you handle dynamic traffic patterns?
VL2 Goals

The Illusion of a Huge L2 Switch

1. L2 semantics

2. Uniform high capacity

3. Performance isolation
Discussion

- What may performance isolation mean?
Objectives in Detail

- **Layer-2 semantics:**
  - Easily assign any server to any service
    - Assigning servers to service should be independent of network topology
  - Configure server with whatever IP address the service expects
  - VM keeps the same IP address even after migration

- **Uniform high capacity:**
  - Maximum rate of server to server traffic flow should be limited only by capacity on network cards

- **Performance isolation:**
  - Traffic of one service should not be affected by traffic of other services (need the above bound)
VL2 Topology: Basic Idea

single-root tree

multi-root tree
The bigger the \( m \), the more flexible in switching.

Q: How big is \( m \) so that each new call can be established w/o moving current calls?

Q: If you can move existing calls, it is only \( m \geq n \).
Folded Clos (Fat-Tree) Topology

Generic K-ary Fat Tree

- K-ary fat tree: three-layer topology (edge, aggregation and core)
  - k pods w/ each pod consisting of 2 layers of k/2 k-port switches
    - each edge switch connects to k/2 servers & k/2 aggr. switches
    - each aggr. switch connects to k/2 edge & k/2 core switches
  - each core switch connects to k pods

**Generic K-ary Fat Tree**

Q: How many servers per pod?
Q: How many servers in total?
Q: How many core switches?
Q: How many links btw each two layers?
Q: How many servers for $k = 48, 96, 144$?
Assume
- Each Int switch has $D_I$ ports;
- Each Aggr has $D_A$ ports

Each Aggr switch uses half ports to connect to TOR switches, half to each Intermediate switch.

Each TOR connects to two Aggr switches.

$D_I$ $D_A/4$ TOR

$20$ (servers)

Each TOR connects $20$ servers.
VL2 Topology

### Table

<table>
<thead>
<tr>
<th>$D=D_I=D_A$ (# of 10G ports)</th>
<th>Max DC size (# of Servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>11,520</td>
</tr>
<tr>
<td>96</td>
<td>46,080</td>
</tr>
<tr>
<td>144</td>
<td>103,680</td>
</tr>
</tbody>
</table>

### Diagram

- **Aggr** nodes connected to **Int** nodes, with **TOR** nodes at the bottom. Each **TOR** node is connected to 20 servers.
Summary: Why Fat-Tree?
Some Other Topologies

Fat-tree [SIGCOMM’08]

Jellyfish (random) [NSDI’12]

BCube [SIGCOMM’10]
Offline Read

Single-Chip “Merchant Silicon” Switches

Image courtesy of Facebook
Multiple switching layers (Why?)

Long cables (fiber)