Duet: Cloud Scale Load Balancing with Hardware and Software

Rohan Gandhi  Hongqiang Harry Liu  Guohan Lu  Jitu Padhye
Y. Charlie Hu  Lihua Yuan  Ming Zhang
Load Balancer is Critical For Online Services

Load balancer provides high availability and scalability

Terminology
VIP – Virtual IP
DIP – Direct IP
Mux – Multiplexer

VIP1, VIP2

Internet

Datacenter

Load balancer (Mux)

VIP1 = 1.1.1.1
VIP2 = 1.1.1.2

DIP = 10.0.1.1
DIP = 10.0.2.2
DIP = 10.0.3.3
DIP = 10.0.4.4

Split VIP traffic to DIPs

bing.com
OneDrive
Existing LBs Have Limitations

Specialized Hardware LBs
- Too costly
  - $100+ million for 15 Tbps
- Poor robustness
  - 1+1 redundancy

Software LBs
- Scale with demand
  - Scale up/down according to VIP traffic
- High robustness
  - n+1 redundancy

Software LBs have cost and performance limitations
Software LB Design

Software LB Benefits:
- High robustness
- Scale with demand

Ananta: Cloud-scale load balancing [SIGCOMM’13]
Software LB has Limitations

High latency inflation: 200 usec
Low capacity: 300k pkts/sec

5k SMuxes needed at 15Tbps traffic in 50k server DC
How can we build high performance, low cost and robust load balancer?

Duet ideas

• Use commodity switches as hardware Muxes
• Use software Muxes as a backstop
Can Switch Act As a Mux

Switches offer:

- High capacity (500+ million pkts/sec)
- Low latency inflation (1 usec)

Mux functionalities

- Split VIP traffic across DIPs
- Forward VIP traffic to DIPs

Switch resources

- ECMP
- Tunneling
Implementing HMux on Switch

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1/32</td>
<td>10.0.0.1, 10.0.0.2</td>
</tr>
<tr>
<td>1.1.1.2/32</td>
<td>20.0.0.1, 20.0.0.2</td>
</tr>
</tbody>
</table>

**Forwarding Table**

<table>
<thead>
<tr>
<th>VIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1/32</td>
</tr>
<tr>
<td>1.1.1.2/32</td>
</tr>
</tbody>
</table>

**ECMP Table**

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

**Tunneling Table**

<table>
<thead>
<tr>
<th>DIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1, 10.0.0.2</td>
</tr>
<tr>
<td>20.0.0.1, 20.0.0.2</td>
</tr>
</tbody>
</table>

**Dest:** VIP   **Src:** Client

**Dest:** DIP   **Src:** HMux

**Dest:** VIP   **Src:** Client
Key Design Challenges

• Limited switch memory
• High failure robustness
• VIP assignment
• VIP migration
Challenge 1: Switches have Limited Memory

Workload: 100k+ VIPs and 1+ millions DIPs
Single HMux cannot store all VIPs and DIPs

Table: Forwarding, ECMP, Tunneling

<table>
<thead>
<tr>
<th>Table</th>
<th>Forwarding</th>
<th>ECMP</th>
<th>Tunneling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. size</td>
<td>16k</td>
<td>4k</td>
<td>512</td>
</tr>
</tbody>
</table>

Max VIPs
Max DIPs
Solution: Partitioning VIPs across HMuxes

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1/32</td>
<td>DIP1, DIP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.2/32</td>
<td>DIP3, DIP4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Capacity</th>
<th>VIPs</th>
<th>DIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single HMux</td>
<td>16k</td>
<td>512</td>
</tr>
<tr>
<td>All HMuxes</td>
<td>16k</td>
<td>512 * 2k = 1M</td>
</tr>
</tbody>
</table>

- Fixed
- Scales with #DIPs
Challenge 2: High Robustness

• Availability during failure?
• Large number of VIPs?
# Idea: Integrate SMux with HMux

<table>
<thead>
<tr>
<th>Feature</th>
<th>HMuxes</th>
<th>SMuxes</th>
<th>Duet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low latency</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>High capacity</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>High availability</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Scale to large #VIPs</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Solution: Use SMuxes As a Backstop

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1/32</td>
<td>DIP1, DIP2</td>
</tr>
</tbody>
</table>

HMux-A

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.1/32</td>
<td>DIP3, DIP4</td>
</tr>
</tbody>
</table>

HMux-B

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.1/32</td>
<td>DIP3, DIP4</td>
</tr>
</tbody>
</table>

SMux-A

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1/32</td>
<td>DIP1, DIP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIP</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.1/32</td>
<td>DIP3, DIP4</td>
</tr>
</tbody>
</table>

SMux-B
Solution: Use SMuxes As a Backstop

- High availability during failure
- Scale to large #VIPs
VIP Traffic Distribution is Highly Skewed

Top 10% VIPs carry 99% traffic

Duet handles 86-99.9% traffic using HMuxes
Challenge 3: How to Assign VIPs?

Objective: Maximize traffic handled by HMuxes

Input:
VIP traffic, DIP locations
Topology

Constraints:
Switch memory
Link capacity
Challenge 4: How to Migrate VIPs?

Current

VIP1
S1
S2
S3
VIP2

Maintain VIP availability

New

VIP2
S1
S2
S3
VIP1

Withdraw and announce

Announce before withdraw

Limited Memory
Solution: Migrate VIPs through SMuxes

Current

Withdraw VIPs from old location

New

Announce VIPs from new location

Fast and maintains VIP availability
Duet Extensions

- SNAT
- Support VIPs with 512+ DIPs
- Port based load balancing
- Load balancing in virtualized networks
Experimental Setup

Testbed

• 10 switches, 3 SMuxes
• 10 VIPs, 34 DIPs

Simulation

• Topology and traffic trace from Azure DC

• High capacity
• High availability
• Low cost
Duet Provides High Capacity

HMux has larger capacity and lower latency than SMux
Duet Provides High Availability

HMux1 fails
38msec
VIP1 traffic falls back SMux

VIP1 on HMux1
VIP2 on HMux2
VIP3 on SMux
Duet reduces cost by 10-24x
Summary

• Specialized and software LBs have cost and performance problems

• **Duet key ideas:**
  - Use commodity switches as HMuxes
  - Use small number of SMuxes as backstop

• **Benefits:**
  - Low latency
  - High capacity
  - High robustness
  - Low cost