Networking Seminar
Stanford University

Madan Jampani
3/12/2015
Can SDN control plane scale without sacrificing abstractions and performance?
Simple yet powerful abstraction

Global Network View

Scaling strong consistency

Managing Complexity
What is the simplest controller?
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
How to improve availability?
How to improve availability?
How to improve availability?

Standby ➔ Primary
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
</tr>
</tbody>
</table>
Why you might need to scale?
Why you might need to scale?
Why you might need to scale?
Why you might need to scale?
Fully Distributed Control Plane
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Multi-Instance</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Abstractions</td>
<td>Performance</td>
<td>Correctness</td>
<td>Availability</td>
<td>Scale</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>-------------</td>
<td>-------------</td>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>Single Instance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Multi Instance</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
“Tell me about your slice?”
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Multi Instance</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
“Tell me about your slice?”
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Multi Instance</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Can we build a complete solution?

Let’s look at prior art...
Distributed Topology Store

Read Topology State

Distributed Topology Store
Cache

Distributed Topology Store

Read Topology State

Cache

Distributed Topology Store
Can we build a solution that meets all criteria?
Events are Switch/Port/Link up/down
Pros

● Simple
● Fast

Cons

● Dropped messages
● Reordered messages
We have a single writer problem
Switch Mastership Terms

We track this in a strongly consistent store
Switch Event Numbers
Partial Ordering of Topology Events

Each event has a unique logical timestamp

(Switch ID, Term Number, Event Number)
To summarize

Each instance has a full copy of network topology

Events are timestamped on arrival and broadcasted

Stale events are dropped on receipt
There is one additional step...

What about dropped messages?
Did you hear about the port that went offline?
Anti-Entropy

Lightweight, Gossip style, peer-to-peer

Quickly bootstraps newly joined nodes
<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Performance</th>
<th>Correctness</th>
<th>Availability</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instance</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Primary/Standby</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Multi Instance</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
A model for state tracking

If you are the observer, Eventual Consistency is the best option

View should be consistent with the network, not with other views
Hiding the complexity

EventuallyConsistentMap<K, V, T>

Plugin your own timestamp generation
What about other Control Plane state...

- Switch to Controller mapping
- Resource Allocations
- Flows
- Various application generated state
What about other Control Plane state...

- Switch to Controller mapping
- Resource Allocations
- Flows
- Various application generated state

In all case strong consistency is either required or highly desirable
Consistency => Coordination

2PC
- All participants need to be available

Consensus
- Only a majority need to participate
State Consistency through Consensus
State Consistency through Consensus
Scaling Consistency
Scaling Consistency
Scaling Consistency
Consistency Cost

- Atomic updates within a shard are “cheap”
- Atomic updates spanning shards use 2PC
Hiding Complexity

ConsistentMap<K, V>
To Conclude

- SDN at scale is possible if we can take care of state management
- Abstractions are important
Thank you