Scale Kubernetes to Support 50,000 Services

Haibin Michael Xie, Senior Staff Engineer/Architect, Huawei
Quinton Hoole, Technical Vice President, Huawei
Agenda

• Challenges while scaling services
• Solutions and prototypes
• Performance data
• Q&A
What are the Challenges while Scaling Services

• Control plane (Master, kubelet, kube-proxy)
  • Deploy services and pods
  • Propagate endpoints

• Data plane (load balancer)
  • Add/remove services in load balancer
  • Accessing services
Control Plane

N nodes per cluster
M pods per second
QPS: N*M endpoints per second
Endpoints

/registry/services/endpoints/default/my-service
/registry/services/specs/default/my-service

```json
{
    "kind": "Endpoints",
    "apiVersion": "v1",
    "metadata": {
        "name": "my-service",
        "namespace": "default",
        "uid": "6b65b3dd-037d-11e7-b2b7-fa163e5e2b3e",
        "creationTimestamp": "2017-03-07T19:18:36Z",
        "enable": true
    },
    "subsets": [
        {
            "addresses": [
                {
                    "ip": "172.17.0.2",
                    "targetRef": {
                        "kind": "Pod",
                        "namespace": "default",
                        "name": "test-9eb8f59f9-61r75",
                        "uid": "54475d42-036a-11e7-b748-fa163e5e2b3e",
                        "resourceVersion": "14078"
                    }
                },
                {
                    "ip": "172.17.0.3",
                    "targetRef": {
                        "kind": "Pod",
                        "namespace": "default",
                        "name": "test-9eb8f59f9-2w5jz",
                        "uid": "54475e58-036a-11e7-b748-fa163e5e2b3e",
                        "resourceVersion": "14051"
                    }
                }
            ],
            "ports": [
                {
                    "port": 9376,
                    "protocol": "TCP"
                }
            ]
        }
    ]
}
```
Control Plane

API Server
- ETCD
  - services
  - pods
  - endpoints

Controller Manager
- Endpoints Controller
- KubeProxy
  - Node

N nodes per cluster
M pods per second
QPS: \(N \times M\) endpoints per second
Load: \(N \times M \times (M+1)/2\) addresses per second
Control Plane Solution

1. Partition endpoints object into multiple objects
   • Pros: reduce Endpoints object size
   • Cons: increase # of objects and requests

2. Central load balancer
   • Pros: reduce connections and requests to API server
   • Cons: one more hop in service routing, require strong HA, limited LB scalability

3. Batch creating/updating endpoints
   • Timer based, no change to data structure in ETCD
   • Pros: reduce QPS
   • Cons: E2E latency is increased by Batch interval
Control Plane Solution

Controller Manager
Endpoints Controller

API Server
ETCD
services
pods
endpoints

KubeProxy
Node

KubeProxy
Node

KubeProxy
Node

N nodes per cluster
M pods per second
QPS: N*M per second
Load: N*M*(M+1)/2 addresses per second

QPS: N per second
Load: N*M addresses per second
Batch Processing Requests Reduction

Test setup:
1 Master, 4 slaves
16 core 2.60GHz, 48GB RAM

One batch per 0.5 second.
➢ QPS: reduced 98%

<table>
<thead>
<tr>
<th>Pods per Service</th>
<th>Number of Service</th>
<th>EndPoints Controller # of Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>Before: 100, After: 10, Reduction: 98.2%</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>Before: 150, After: 14, Reduction: 98.2%</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>Before: 200, After: 17, Reduction: 98.5%</td>
</tr>
</tbody>
</table>

# of Requests

![Graph showing the number of requests vs. latency]
### Batch Processing E2E Latency Reduction

**Latency:** reduced 60+%

<table>
<thead>
<tr>
<th>Pods per Service</th>
<th>Number of Service</th>
<th>E2E Latency (Second)</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8.5</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>13.5</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>22.8</td>
<td>7.8</td>
</tr>
</tbody>
</table>

---

**Graph:**

- **E2E Latency**
  - X-axis: Number of Services (Each service has 10 pods)
  - Y-axis: E2E Latency (seconds)
  - Blue line: Base Code
  - Red line: Optimized
• What is IPTables?
  • iptables is a user-space application that allows configuring Linux kernel firewall (implemented on top of Netfilter) by configuring chains and rules.
  • What is Netfilter? A framework provided by the Linux kernel that allows customization of networking-related operations, such as packet filtering, NAT, port translation etc.

• Issues with IPTables as load balancer
  • Latency to access service (routing latency)
  • Latency to add/remove rule
IPTables Example

```
# Iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target prot opt source               destination
  KUBE-SERVICES all -- anywhere anywhere /* kubernetes service portals */ 1
  DOCKER all -- anywhere anywhere ADDRTYPE match dst-type LOCAL

Chain KUBE-SEP-G3MLSGWLUPEIMXS (1 references) 4
target prot opt source               destination
  MARK all -- 172.16.16.2 anywhere /* default/webpod-service: */ MARK set 0x4d415351
  DNAT tcp -- anywhere anywhere /* default/webpod-service: */ tcp:172.16.16.2:80

Chain KUBE-SEP-OUBP2XSUG3G4CYYB (1 references) 4
target prot opt source               destination
  MARK all -- 192.168.190.128 anywhere /* default/kubernetes: */ MARK set 0x4d415351
  DNAT tcp -- anywhere anywhere /* default/kubernetes: */ tcp:192.168.190.128:6443

Chain KUBE-SEP-PXEMGP3B44XONJEO (1 references) 4
target prot opt source               destination
  MARK all -- 172.16.91.2 anywhere /* default/webpod-service: */ MARK set 0x4d415351
  DNAT tcp -- anywhere anywhere /* default/webpod-service: */ tcp:172.16.91.2:80

Chain KUBE-SERVICES (2 references) 2
target prot opt source               destination
  KUBE-SVC-N4RX4VPNP4ATLCGG tcp -- anywhere 192.168.3.237 /* default/webpod-service: cluster IP */ tcp dpt:http
  KUBE-SVC-6N4SJQIF3IX3FORG tcp -- anywhere 192.168.3.1 /* default/kubernetes: cluster IP */ tcp dpt:https
  KUBE-NODEPORTS all -- anywhere anywhere /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */ ADDRTYPE match dst-type LOCAL

Chain KUBE-SEP-OUBP2XSUG3G4CYYB (1 references)
  target prot opt source               destination
  KUBE-SEP-G3MLSGWLUPEIMXS all -- anywhere anywhere /* default/webpod-service: */ ADDRTYPE match dst-type LOCAL
  KUBE-SEP-PXEMGP3B44XONJEO all -- anywhere anywhere /* default/webpod-service: */
```

IPTables Service Routing Performance

Where is latency generated?

- Long list of rules in a chain
- Enumerate through the list to find a service and pod

In this test, there is one entry per service in KUBE-SERVICES chain.
Latency to Add IPTables Rules

• Where is the latency generated?
  • not incremental
  • copy all rules
  • make changes
  • save all rules back
  • IPTables locked during rule update

• Time spent to add one rule when there are 5k services (40k rules): 11 minutes

• 20k services (160k rules): 5 hours
Data Plane Solution

- Re-struct IPTables using search tree (Performance benefit)
- Replace IPTables with IPVS (Performance and beyond)
Restruct IPTables by Search Tree

Service VIP range: 10.10.0.0/16  
CIDR list = [16, 24], defines tree layout  
Create 3 services: 10.10.1.5, 10.10.1.100, 10.10.100.1

Search tree based service routing time complexity: $O(\sqrt{n})$, $m$ is tree depth

Original service routing time complexity: $O(n)$
What is IPVS

• Transport layer load balancer which directs requests for TCP and UDP based services to real servers.
• Same to IPTables, IPVS is built on top of Netfilter.
• Support 3 load balancing mode: NAT, DR and IP Tunneling.
IPVS vs. IPTables

IPTables:
• Operates tables provided by linux firewall
• IPTables is more flexible to manipulate package at different stage: Pre-routing, post-routing, forward, input, output.
• IPTables has more operations: SNAT, DNAT, reject packets, port translation etc.

Why using IPVS?
• Better performance (Hashing vs. Chain)
• More load balancing algorithm
  • Round robin, source/destination hashing.
  • Based on least load, least connection or locality, can assign weight to server.
• Support server health check and connection retry
• Support sticky session
IPVS Load Balancing Mode in Kubernetes

- Not public released yet
- No Kubernetes behavior change, complete functionalities: external IP, nodePort etc
- Kube-proxy startup parameter mode=IPVS, in addition to original modes: mode=userspace and mode=iptables
- Kube-proxy lines of code: 11800
- IPVS mode adds 680 lines of code, dependent on seasaw library
IPVS vs. IPTables Latency to Add Rules

Measured by iptables and ipvsadm, observations:

- In IPTables mode, latency to add rule increases significantly when # of service increases
- In IPVS mode, latency to add VIP and backend IPs does not increase when # of service increases

<table>
<thead>
<tr>
<th># of Services</th>
<th>1</th>
<th>5,000</th>
<th>20,000</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Rules</td>
<td>8</td>
<td>40,000</td>
<td>160,000</td>
</tr>
<tr>
<td>IPTables</td>
<td>2 ms</td>
<td>11 min</td>
<td>5 hours</td>
</tr>
<tr>
<td>IPVS</td>
<td>2 ms</td>
<td>2 ms</td>
<td>2 ms</td>
</tr>
</tbody>
</table>
IPVS vs. IPTables Network Bandwidth

- Measured by qperf
- Bandwidth, QPS, Latency have similar pattern
- Env: 1 master, 4 slaves, 8 pods, all services use these 8 pods
- Each service exposes 4 ports (4 entries in KUBE-SERVICES chain)

### Bandwidth

<table>
<thead>
<tr>
<th># of services</th>
<th>1</th>
<th>1000</th>
<th>1000</th>
<th>5000</th>
<th>5000</th>
<th>10000</th>
<th>10000</th>
<th>25000</th>
<th>25000</th>
<th>50000</th>
<th>50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth, IPTables (MB/S)</td>
<td>66.6</td>
<td>64</td>
<td>56</td>
<td>50</td>
<td>38.6</td>
<td>15</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bandwidth, IPVS (MB/S)</td>
<td>65.3</td>
<td>61.7</td>
<td>55.3</td>
<td>53.5</td>
<td>53.8</td>
<td>43</td>
<td>43.5</td>
<td>30</td>
<td>28.5</td>
<td>24</td>
<td>23.8</td>
</tr>
</tbody>
</table>
More Perf/Scalability Work Done

• Scale nodes and pods in single cluster
• Reduce E2E latency of deploying pods/services
• Increase pod deployment throughput
• Improve scheduling performance
Thank You

quinton.hoole@huawei.com
haibin.michael.xie@huawei.com