Scale Kubernetes to Support 50,000 Services

Haibin Michael Xie, Senior Staff Engineer/Architect, Huawei

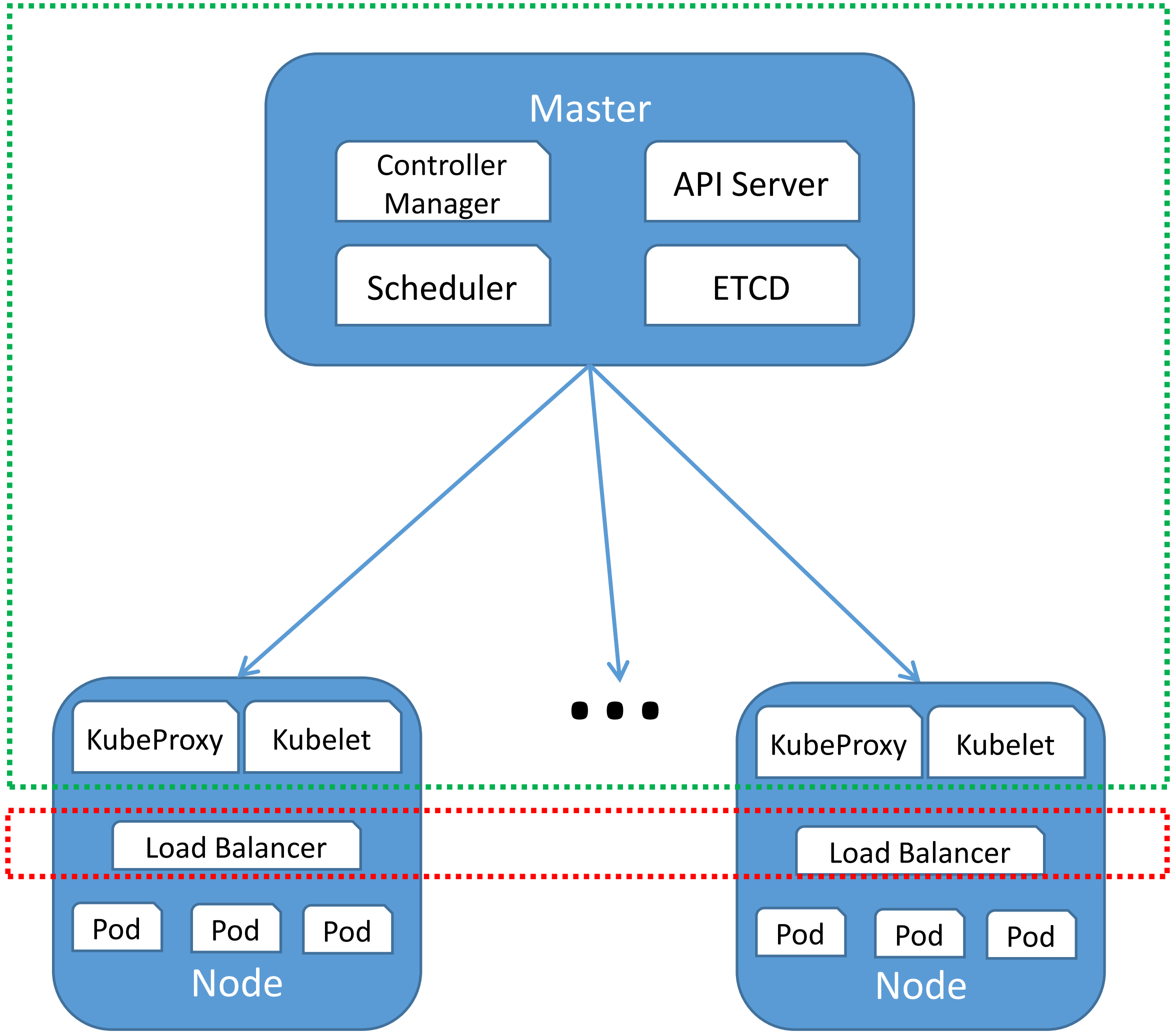
Agenda

- Challenges while scaling services
- Solutions and prototypes
- Performance data
- Q&A

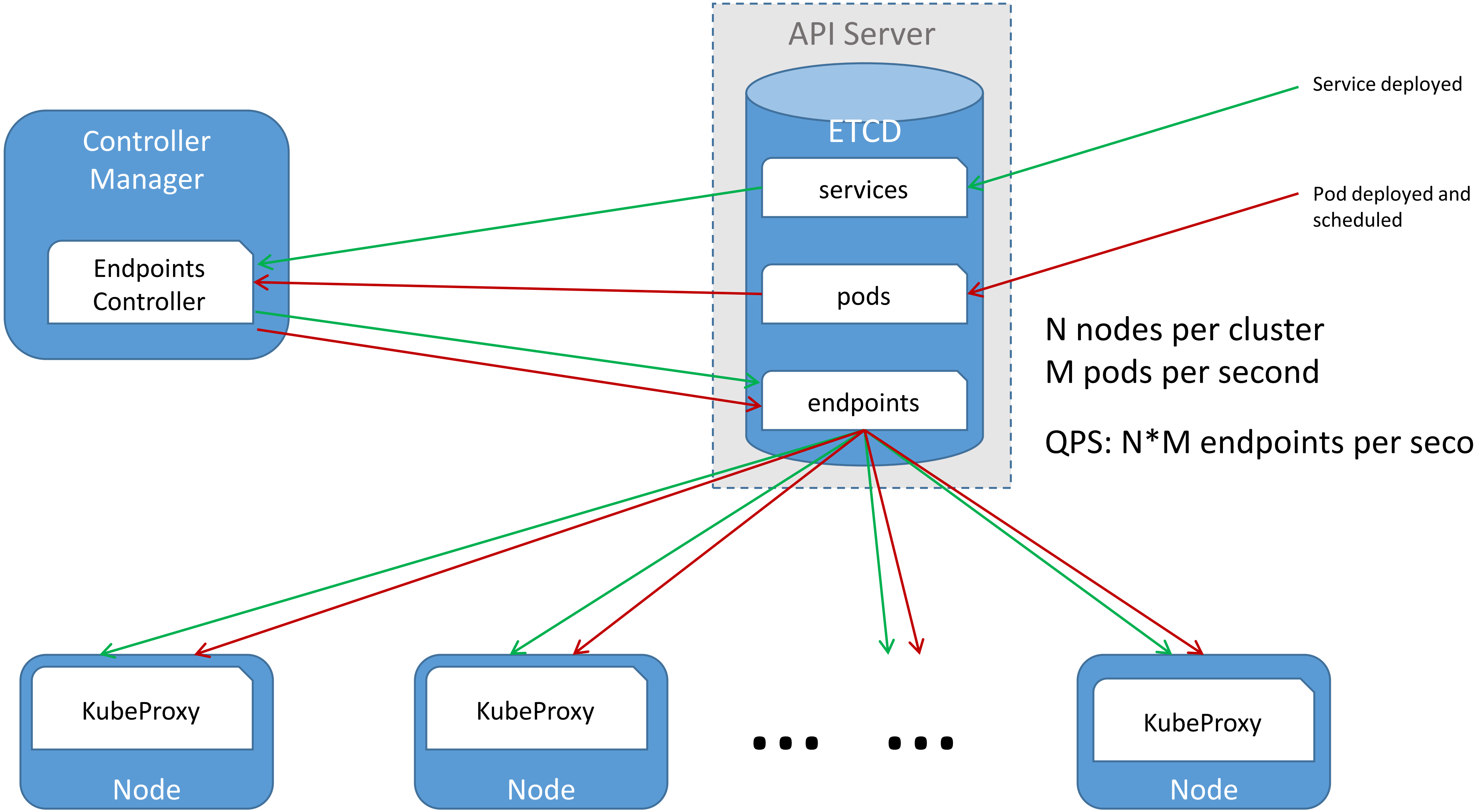
What are the Challenges while Scaling Services

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

- Data plane (load balancer)
  - Add/remove services in load balancer
  - Propagate endpoints



Control Plane



N nodes per cluster
M pods per second

QPS: N*M endpoints per second

Endpoints

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

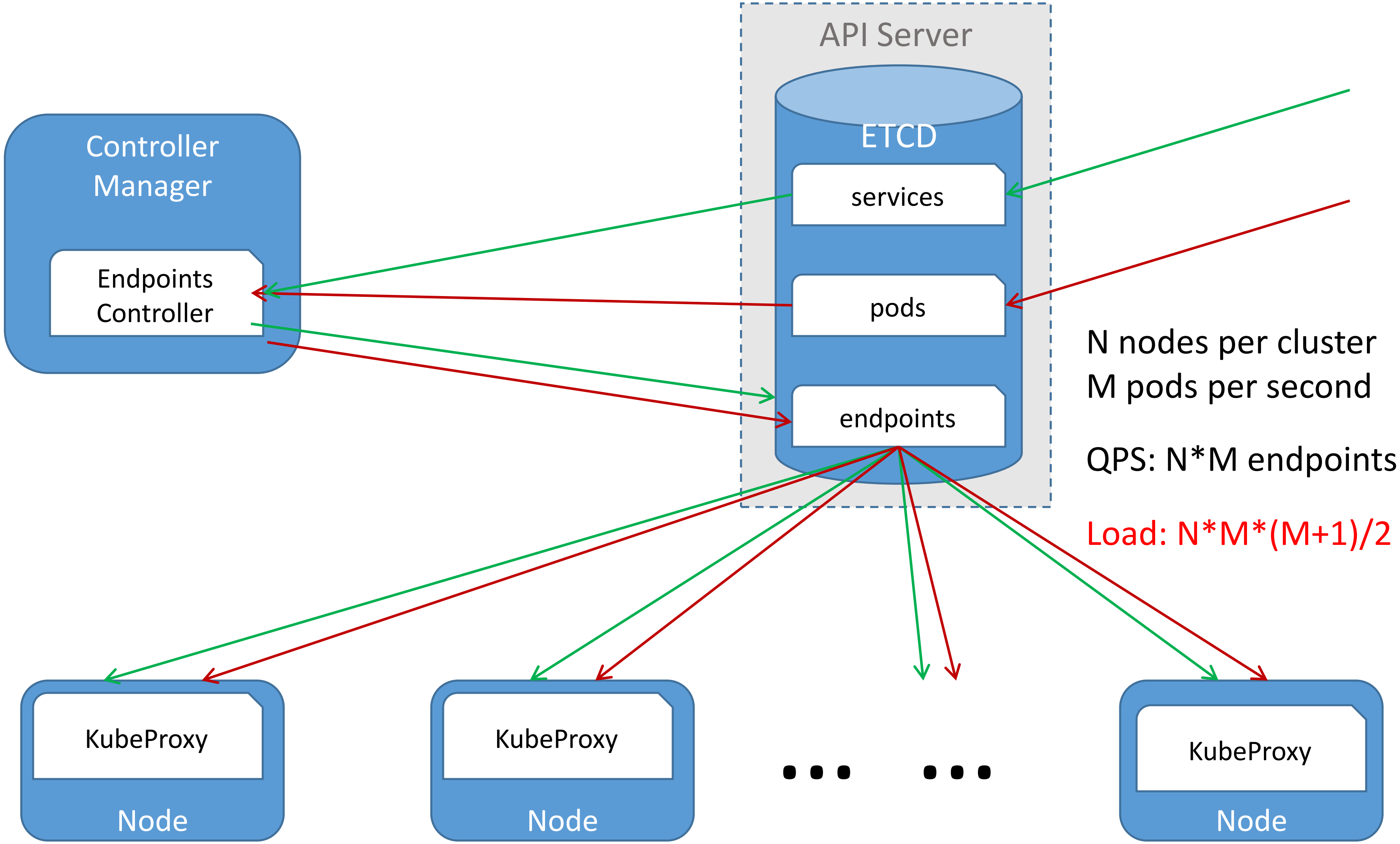
/registry/services/specs/default/my-service


```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-service",
    "namespace": "default",
    "uid": "6ba5bdd2-037d-11e7-b2b7-fa163e5e2b3e",
    "creationTimestamp": "2017-03-07T21:31:26Z",
    "enable": true
  },
  "spec": {
    "ports": [
      {
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376
      }
    ],
    "selector": {
      "app": "nginx"
    },
    "clusterIP": "10.10.10.104",
    "type": "ClusterIP",
    "sessionAffinity": "None"
  },
  "status": {
    "loadBalancer": {}
  }
}
```



```
{
  "kind": "Endpoints",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-service",
    "namespace": "default",
    "uid": "dcf04517-036a-11e7-b748-fa163e5e2b3e",
    "creationTimestamp": "2017-03-07T19:18:36Z",
    "enable": true
  },
  "subsets": [
    {
      "addresses": [
        {
          "ip": "172.17.0.2",
          "targetRef": {
            "kind": "Pod",
            "namespace": "default",
            "name": "test-968485994-61r75",
            "uid": "54475d42-036a-11e7-b748-fa163e5e2b3e",
            "resourceVersion": "14070"
          }
        },
        {
          "ip": "172.17.0.3",
          "targetRef": {
            "kind": "Pod",
            "namespace": "default",
            "name": "test-968485994-2w5jz",
            "uid": "54475e58-036a-11e7-b748-fa163e5e2b3e",
            "resourceVersion": "14051"
          }
        }
      ],
      "ports": [
        {
          "port": 9376,
          "protocol": "TCP"
        }
      ]
    }
  ]
}
```

Control Plane



N nodes per cluster
M pods per second

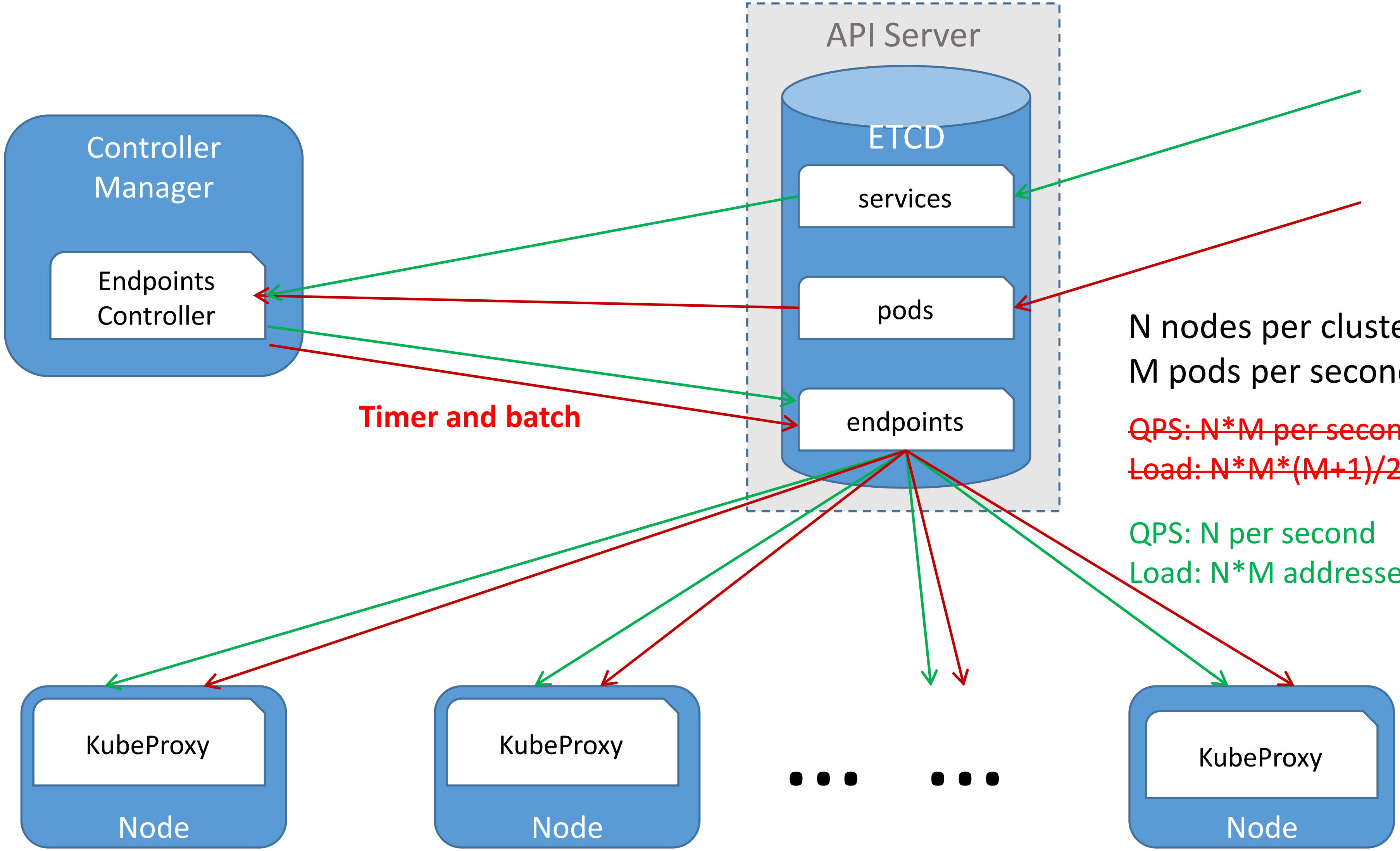
QPS: N*M endpoints per second

Load: N*M*(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



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:

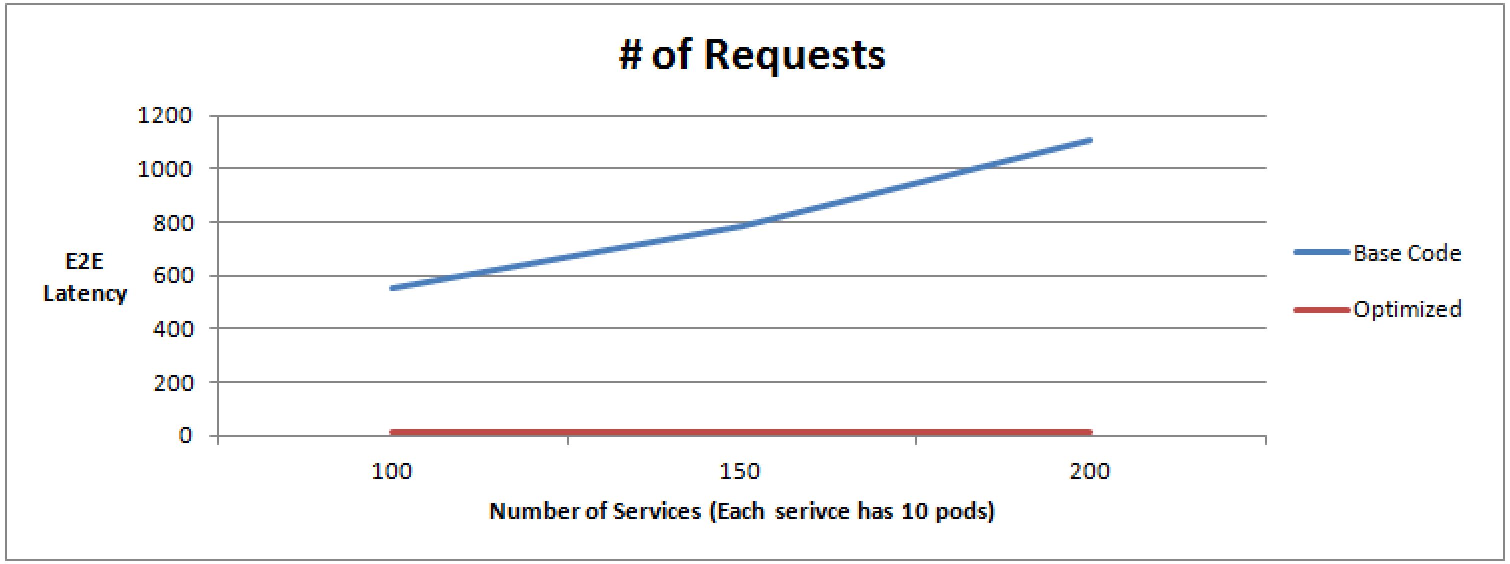
1 Master， 4 slaves

16 core 2.60GHz, 48GB RAM

One batch per 0.5 second.

- QPS： **reduced 98%**

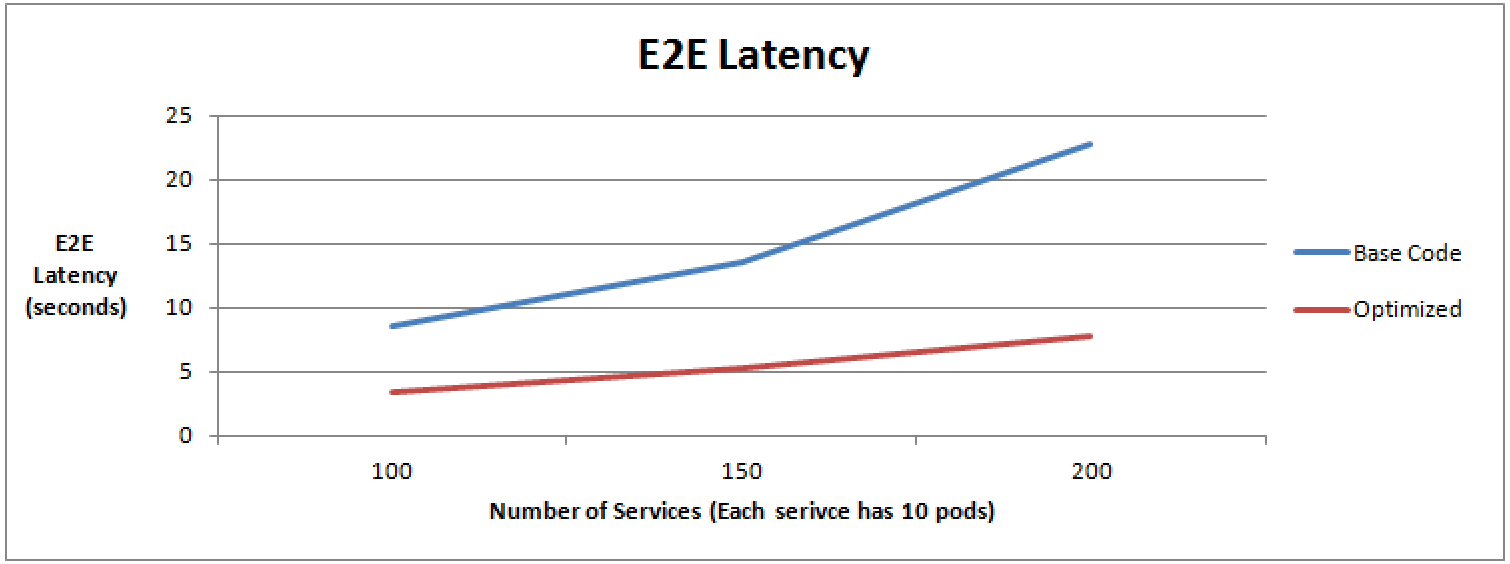
Pods per Service	Number of Service	EndPoints Controller # of Requests		
		Before	After	Reduction
10	100	551	10	98.2%
	150	785	14	98.2%
	200	1105	17	98.5%



Batch Processing E2E Latency Reduction

Latency: **reduced 60+%**

Pods per Service	Number of Service	E2E Latency (Second)		
		Before	After	Reduction
10	100	8.5	3.5	59.1%
	150	13.5	5.3	60.9%
	200	22.8	7.8	65.8%



Data Panel

- 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

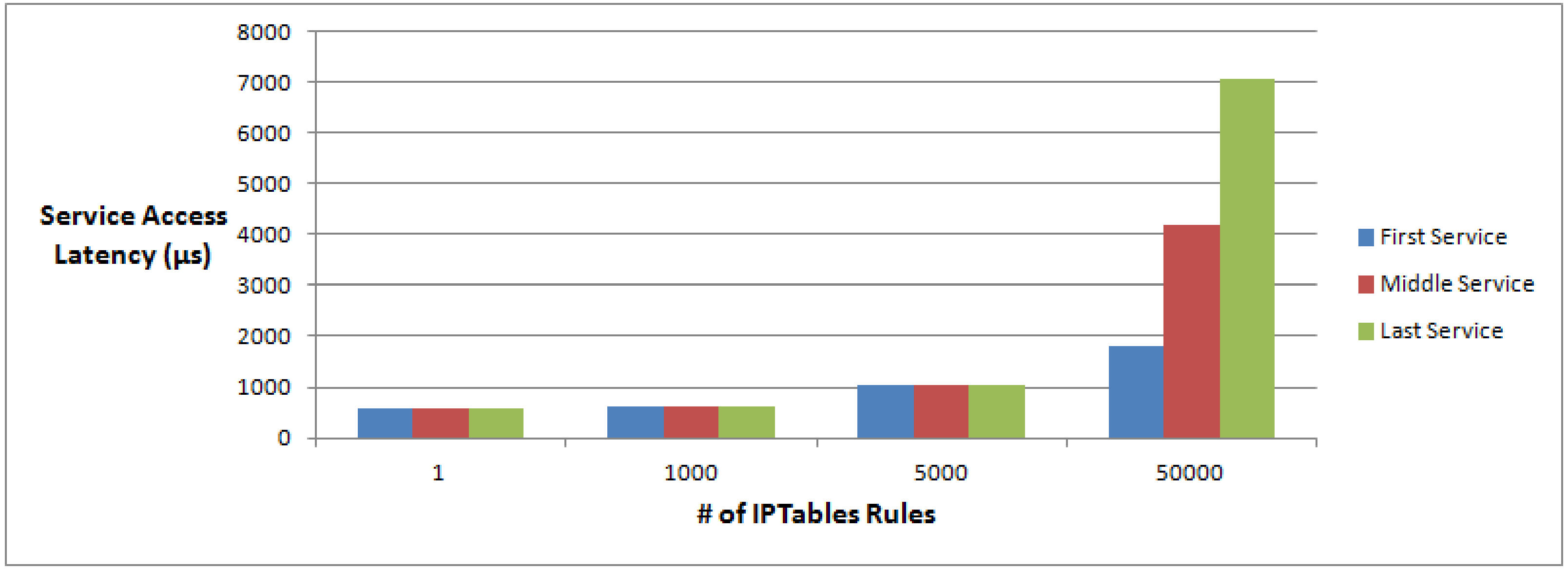
```
#   Iptables –t nat –L –n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
KUBE-SERVICES  all  --  anywhere             anywhere             /* kubernetes service portals */ ← 1
DOCKER     all  --  anywhere             anywhere             ADDRTYPE match dst-type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

Chain KUBE-SEP-G3MLSGWVLUPEIMXS (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 to:172.16.16.2:80

Chain KUBE-SEP-OUBP2X5UG3G4CYYB (1 references)
target     prot opt source               destination
MARK       all  --  192.168.190.128      anywhere             /* default/kubernetes: */ MARK set 0x4d415351
DNAT       tcp  --  anywhere             anywhere             /* default/kubernetes: */ tcp to: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 to: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-SVC-6N4SJQIF3IX3FORG (1 references)
target     prot opt source               destination
KUBE-SEP-OUBP2X5UG3G4CYYB  all  --  anywhere             anywhere             /* default/kubernetes: */

Chain KUBE-SVC-N4RX4VPNP4ATLCGG (1 references) ← 3
target     prot opt source               destination
KUBE-SEP-G3MLSGWVLUPEIMXS  all  --  anywhere             anywhere             /* default/webpod-service: */ statistic mode random probability 0.50000000000
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.



	1 Service (μs)	1000 Services (μs)	10000 Services (μs)	50000 Services (μs)
First Service	575	614	1023	1821
Middle Service	575	602	1048	4174
Last Service	575	631	1050	7077

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

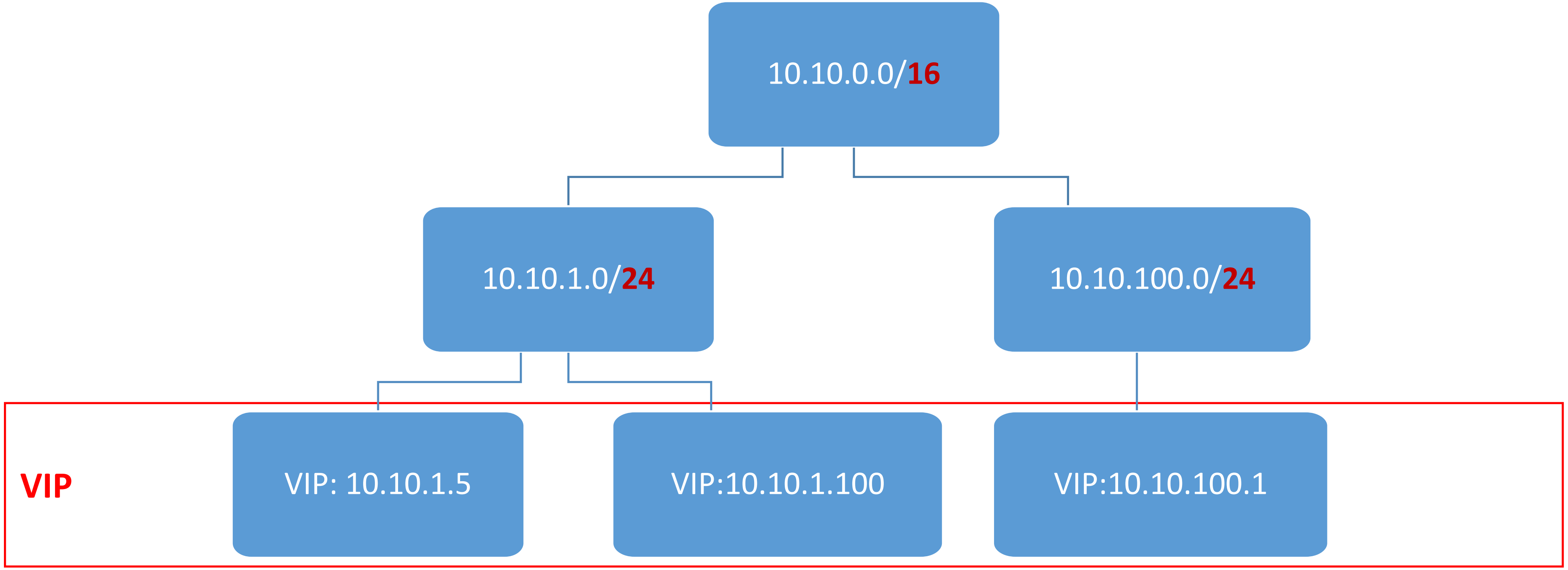
- Re-struct IPTables using search tree (Performance benefit)
- Replace IPTables with IPVS (Performance and beyond)

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: $O(\sqrt[m]{n})$ **, 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

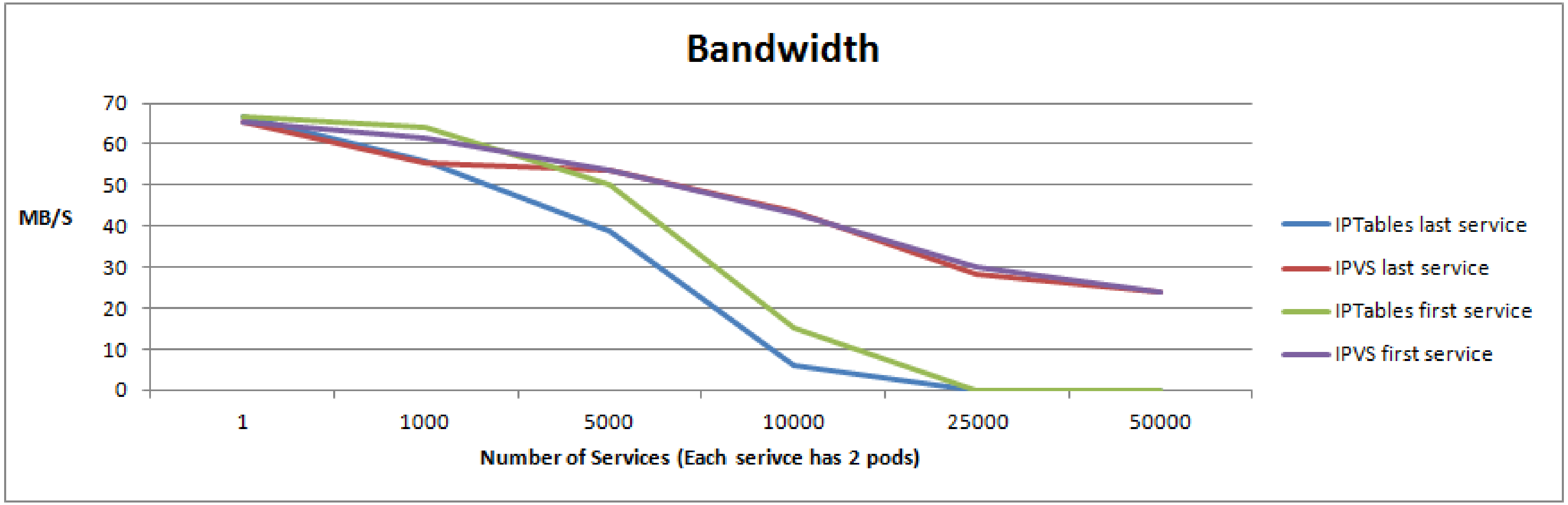
Measured by iptables and ipvsadm, observations:

- 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

# of Services	**1**	**5,000**	**20,000**
# of Rules	**8**	**40,000**	**160,000**
IPTables	2 ms	11 min	5 hours
IPVS	2 ms	2 ms	2 ms

IPVS vs. IPTables Network Bandwidth

- Measured by qperf
- Each service exposes 4 ports (4 entries in KUBE-SERVICES chain)
- Bandwidth, QPS, Latency have similar pattern



ith service	first	first	last	first	last	first	last	first	last	first	last
# of services	**1**	**1000**	**1000**	**5000**	**5000**	**10000**	**10000**	**25000**	**25000**	**50000**	**50000**
Bandwidth, IPTables (MB/S)	66.6	64	56	50	38.6	15	6	0	0	0	0
Bandwidth, IPVS (MB/S)	65.3	61.7	55.3	53.5	53.8	43	43.5	30	28.5	24	23.8

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

haibin.michael.xie@huawei.com