clusterIP建联超1s问题

Subtitle

2022/10/05

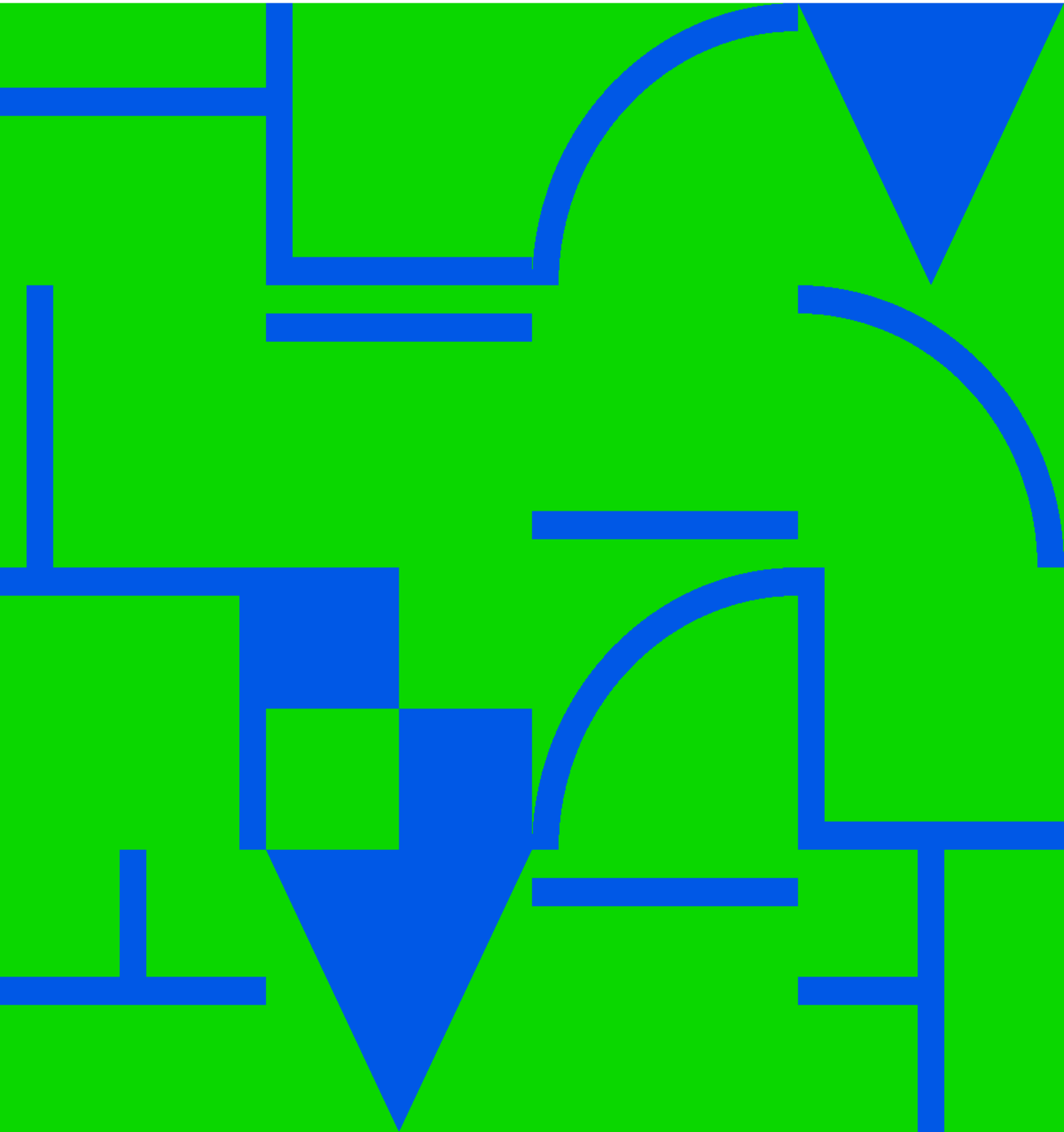


Table of Contents

clusterIP建联超1s问题

最终结论

connreusemode改为0后，通过ingress-nginx调用改为通过clusterIP调用，但是不用长连接，平均响应时间还不如（增加1ms□ingress-nginx代理（ingress-nginx到upstream是长连接）。

应尽量使用长连接。

结论 单纯docker环境中，容器内部访问外部网络，同样需要通过iptables做SNAT，但是测试并未发现性能问题。因此ipvs的问题最终发现是因为net.ipv4.vs.connreusemode参数设置为1导致的。设置为0，对端口重用不做任何特殊处理，则没有性能问题

- 调整connreusemode参数：https://github.com/kubernetes/kubernetes/issues/70747#issuecomment-437558941
- 调整connreusemode的代价：https://github.com/moby/moby/issues/35082#issuecomment-419072026
- ~~ipvs使用iptables做SNAT，受限于可用端口数量，吞吐会受到严重影响~~。参考：http://zh.linuxvirtualserver.org/node/294

conn_reuse_mode - INTEGER
1 - default

Controls how ipvs will deal with connections that are detected
port reuse. It is a bitmap, with the values being:

0: disable any special handling on port reuse. The new
connection will be delivered to the same real server that was
servicing the previous connection. This will effectively
disable expire_nodest_conn.

bit 1: enable rescheduling of new connections when it is safe.
That is, whenever expire_nodest_conn and for TCP sockets, when

the connection is in TIME_WAIT state (which is only possible if
you use NAT mode).

bit 2: it is bit 1 plus, for TCP connections, when connections
are in FIN_WAIT state, as this is the last state seen by load
balancer in Direct Routing mode. This bit helps on adding new
real servers to a very busy cluster.

现象

```
for id in `seq 1 100`;do curl -s "http://169.169.93.8" -o /dev/null -w "%{time_connect} -
%{time_total}\n" ;done |awk '$3>0.1'
```

难以复现，可能跟当时网络状况有关

UPDATE20181104 已复现

容器内部

```
dd4ddd9f6-b2ptr:~# for id in `seq 1 100`;do curl -s "http://169.169.218.191" -o /dev/null -w
"%{time_connect} %{time_total}
\n";done |awk '$2>0.1' |wc -l
13
```

宿主机

```
[root@k8s-node-1 ~]# for id in `seq 1 100`;do curl -s "http://169.169.218.191" -o /dev/null -w
"%{time_connect} %{time_total}\n";done |awk '$2>0.1' |wc -l
0
[root@k8s-node-1 ~]# for id in `seq 1 100`;do curl -s "http://169.169.218.191" -o /dev/null -w
"%{time_connect} %{time_total}\n";done |awk '$2>0.1' |wc -l
0
```

容器内部直连pod ip，需要加上–connect-timeout参数，否则将尝试30s以上

```
dd4ddd9f6-b2ptr:~# for id in `seq 1 100`;do curl --connect-timeout 1 -s "http://172.20.17.50" -o
/dev/null -w "%{time_conne
ct} %{time_total}\n";done |awk '$2>0.1' |wc -l
16
```

宿主机直连pod ip

```
[root@k8s-node-1 ~]# for id in `seq 1 100`;do curl --connect-timeout 1 -s "http://172.20.17.50" -o
/dev/null -w "%{time_connect} %{time_total}\n";done |awk '$2>0.1' |wc -l
0
```

可以看到不论是clusterIP还是pod ip，只要到了容器里面，就变的很慢（是因为并发和时间不够，没有达到宿主机的local port range定义的数量，后面用ab压测，只要是连clusterIP，容器内外都有问题）

hairpin mode的关系？

kubelet日志

```
W1015 15:36:28.344597   26031 docker_service.go:545] Hairpin mode set to "promiscuous-bridge" but kubenet is not enabled, falling back to "hairpin-veth"
I1015 15:36:28.345008   26031 docker_service.go:238] Hairpin mode set to "hairpin-veth"
```

根据官方文档hairpin-veth模式会修改网卡的配置：

If the effective hairpin mode is hairpin-veth, ensure the Kubelet has the permission to operate in /sys on node. If everything works properly, you should see something like:

```
for intf in /sys/devices/virtual/net/cbr0/brif/*; do cat $intf/hairpin_mode; done
1
1
1
1
```

但是使用calico时，/sys/devices/virtual/net/tunl0/下没有brif文件夹。通过以下issue:
https://github.com/kubernetes/kubernetes/issues/45790#issuecomment-302539755

> There's no special configuration required when using a veth without a bridge (e.g. calico, p2p). Traffic hits the kube-proxy's Service DNAT rule, and is routed back to the Pod IP, and then gets masqueraded.

hairpin mode主要涉及容器内如无法访问自己的clusterIP的问题，和这里的网络质量应该没关系

IPVS问题

线索：https://github.com/moby/moby/issues/35082#issuecomment-340515934

```
TCP  169.169.218.191:80 rr
  -> 172.20.2.48:80           Masq   1    0      1952
  -> 172.20.10.56:80          Masq   1    0      1752
  -> 172.20.16.45:80          Masq   1    1      1977
  -> 172.20.17.50:80          Masq   1    0      1817
  -> 172.20.21.59:80          Masq   1    0      1744
  -> 172.20.31.149:80         Masq   1    2      1732
  -> 172.20.37.19:80          Masq   1    1      1704
  -> 172.20.38.35:80          Masq   1    0      1697
  -> 172.20.39.71:80          Masq   1    0      1700
  -> 172.20.40.47:80          Masq   1    0      1739
  -> 172.20.41.47:80          Masq   1    0      1734
  -> 172.20.42.67:80          Masq   1    0      1725
  -> 172.20.44.50:80          Masq   1    0      1684
```

内部端口不够用: https://github.com/moby/moby/issues/35082#issuecomment-382397654

```
domain-api-6fc6d88b8b-6zjtp:~# netstat -ant |grep "169.169.218.191" |wc -l
17926
domain-api-6fc6d88b8b-6zjtp:~# sysctl -a |grep "local_port_range"
net.ipv4.ip_local_port_range = 32768    61000
domain-api-6fc6d88b8b-6zjtp:~# exit
[root@k8s-node-1 ~]# sysctl -a |grep "local_port_range"
net.ipv4.ip_local_port_range = 2000 65000
```

手动创建一个deployment，通过securityContext设置net.ipv4.ip_local_port_range 为2000 2002,经测试，无论是clusterIP还是外部域名，均只能成功连接3次，并且当端口用尽时，很快就返回了（线上表现为卡顿很久），并且提示无可用端口

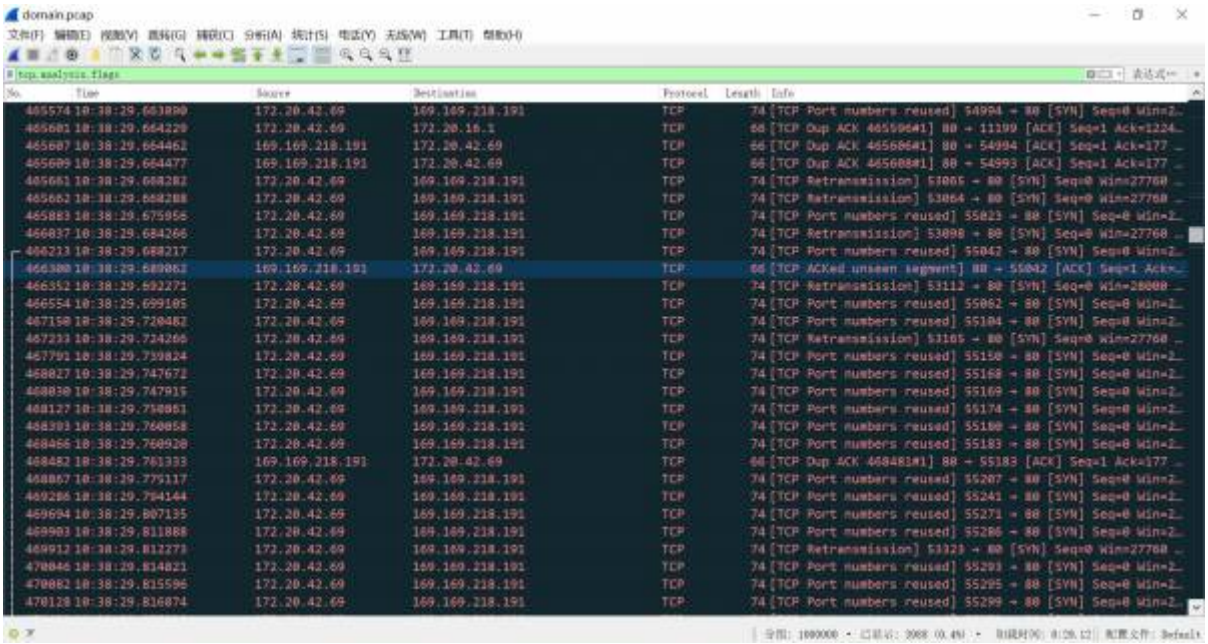
```
* Rebuilt URL to: http://169.169.218.191/
*   Trying 169.169.218.191...
* TCP_NODELAY set
* Immediate connect fail for 169.169.218.191: Address not available
* Closing connection 0
0.000000 0.000000
```

由此可见，此问题不一定是本地端口范围问题

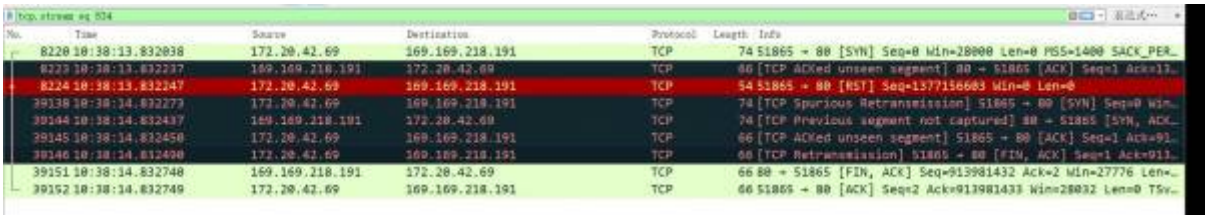
ipvs

- https://github.com/cloudnativelabs/kube-router/issues/544
- https://github.com/cloudnativelabs/kube-router/issues/544#issuecomment-430682391
- https://github.com/cloudnativelabs/kube-router/issues/544#issuecomment-431425219
- https://tech.xing.com/a-reason-for-unexplained-connection-timeouts-on-kubernetes-docker-abd041cf7e02
- https://serverfault.com/questions/558234/tcp-port-numbers-reused-and-tcp-retransmission

抓包



一个失败的请求



压测

- ab压测报告
- https://github.com/kubernetes/kubernetes/issues/65820
- https://kubernetes.io/zh/blog/2018/07/09/ipvs-based-in-cluster-load-balancing-deep-dive/

结论

~~连接clusterIP时使用长连接~~

复现

非容器环境下复现，变成纯粹的ipvs问题.

创建一个虚拟ip 192.168.1.100, 通过ipvs转发到一个部署了nginx的机器上，然后在该机器上分别直接压测nginx和通过vip压测，对比性能

创建虚拟网卡

首先创建虚拟网卡，参考
https://unix.stackexchange.com/questions/152331/how-can-i-create-a-virtual-ethernet-interface-on-a-machine-without-a-physical-ad

```
# modprobe dummy
# ip link set name dummy0 dev dummy0
# ip link show dummy0
33: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT
    link/ether da:7c:6f:7f:43:a4 brd ff:ff:ff:ff:ff:ff

# ip link add name ipvs0 type dummy
# ip link show ipvs0
34: ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT
    link/ether 6a:0b:d9:13:53:4d brd ff:ff:ff:ff:ff:ff
# ip addr add 192.168.1.100/32 dev ipvs0
# ip addr show ipvs0
34: ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noqueue state DOWN
    link/ether 6a:0b:d9:13:53:4d brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/32 scope global ipvs0
       valid_lft forever preferred_lft forever
```

ipvs配置

参考：https://www.cnblogs.com/liwei0526vip/p/6370103.html

首先修改 ipvs conn*tab*bits参数之后在加载ipvs模块

```
# cat /etc/modprobe.d/ip_vs.conf
options ip_vs conn_tab_bits=20
# modprobe ip_vs
```

创建vip

```
# ipvsadm -A -t 192.168.1.100:80 -s rr
# ipvsadm -a -t 192.168.1.100:80 -r 10.112.35.99:80 -m -w 1
# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=1048576)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.100:80 rr
  -> 10.112.35.99:80              Masq    1      0          0

# 10.112.35.99上需添加vip的路由
route add -host 192.168.1.100/32 gw 配置vip的机器ip
```

-C 清除表中所有的记录
-A --add-service在服务器列表中新添加一条新的虚拟服务器记录
-t 表示为tcp服务
-u 表示为udp服务
-s --scheduler 使用的调度算法， rr | wrr | lc | wlc | lblb | lblcr | dh | sh | sed | nq 默认调度算法是 wlc
ipvsadm -a -t 192.168.3.187:80 -r 192.168.200.10:80 -m -w 1
-a --add-server 在服务器表中添加一条新的真实主机记录
-t --tcp-service 说明虚拟服务器提供tcp服务
-u --udp-service 说明虚拟服务器提供udp服务
-r --real-server 真实服务器地址
-m --masquerading 指定LVS工作模式为NAT模式
-w --weight 真实服务器的权值
-g --gatewaying 指定LVS工作模式为直接路由器模式（也是LVS默认的模式）
-i --ipip 指定LVS的工作模式为隧道模式
-p 会话保持时间，定义流量呗转到同一个realserver的会话存留时间

参
见：https://mritd.me/2017/10/10/kube-proxy-use-ipvs-on-kubernetes-1.8/

重点说一下 –masquerade-all 选项: kube-proxy ipvs 是基于 NAT 实现的，当创建一个 service 后，kubernetes 会在每个节点上创建一个网卡，同时帮你将 Service IP(VIP) 绑定上，此时相当于每个 Node 都是一个 ds，而其他任何

Node 上的 Pod，甚至是宿主机服务(比如 kube-apiserver 的 6443)都可能成为 rs；按照正常的 lvs nat 模型，所有 rs 应该将 ds 设置成为默认网关，以便数据包在返回时能被 ds 正确修改；在 kubernetes 将 vip 设置到每个 Node 后，默认路由显然不可行，所以要设置 –masquerade-all 选项，以便反向数据包能通过

压测

```
# ab -c100 -t300 -n20000000 -r http://192.168.1.100/b
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.100 (be patient)


Completed 2000000 requests
Finished 3444742 requests


Server Software:        nginx/1.10.2
Server Hostname:        192.168.1.100
Server Port:            80

Document Path:          /b
Document Length:        3650 bytes

Concurrency Level:      100
Time taken for tests:   300.003 seconds
Complete requests:      3444742
Failed requests:        0
Write errors:           0
Non-2xx responses:      3444791
Total transferred:      13169435993 bytes
HTML transferred:       12573487150 bytes
Requests per second:    11482.37 [#/sec] (mean)
Time per request:       8.709 [ms] (mean)
Time per request:       0.087 [ms] (mean, across all concurrent requests)
Transfer rate:          42868.89 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4  3.5      4    1008
Processing:     0    5  3.7      4     221
Waiting:        0    4  1.8      4     210
Total:          1    9  5.1      8    1021

Percentage of the requests served within a certain time (ms)
  50%      8
  66%      9
  75%     10
```

```
 80%    10
 90%    11
 95%    13
 98%    17
 99%    19
100%  1021 (longest request)
```

结果显示性能完全正常

conn_tab_bits=12

修改conn*tab*bits为默认的12

```
# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.100:80 rr
  -> 10.112.35.99:80              Masq    1      0          1
```

压测

```
# ab -c100 -t300 -n20000000 -r http://192.168.1.100/b
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.100 (be patient)
Completed 2000000 requests
Finished 2991631 requests


Server Software:        nginx/1.10.2
Server Hostname:        192.168.1.100
Server Port:            80

Document Path:          /b
Document Length:        3650 bytes

Concurrency Level:      100
Time taken for tests:   300.002 seconds
Complete requests:      2991631
Failed requests:        0
Write errors:           0
Non-2xx responses:      2991659
Total transferred:      11437112357 bytes
HTML transferred:       10919555350 bytes
Requests per second:    9972.03 [#/sec] (mean)
Time per request:       10.028 [ms] (mean)
Time per request:       0.100 [ms] (mean, across all concurrent requests)
Transfer rate:          37229.89 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4  4.4      4    1008
Processing:     1    6  3.6      5     234
Waiting:        1    5  1.9      4     207
Total:          2   10  5.7     10    1018

Percentage of the requests served within a certain time (ms)
  50%     10
  66%     11
  75%     11
  80%     12
  90%     14
  95%     16
  98%     19
  99%     20
 100%   1018 (longest request)
```

和conn*tab*bits=20时差不多

net.ipv4.vs.conntrack

对比内核参数，kubernetes里net.ipv4.vs.conntrack为1，手动安装的为0，改为1在压测一次

```
# ab -c100 -t300 -n20000000 -r http://192.168.1.100/b
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.100 (be patient)
Completed 2000000 requests
Finished 2995789 requests


Server Software:        nginx/1.10.2
Server Hostname:        192.168.1.100
Server Port:            80

Document Path:          /b
Document Length:        3650 bytes

Concurrency Level:      100
Time taken for tests:   300.000 seconds
Complete requests:      2995789
Failed requests:        1
   (Connect: 1, Receive: 0, Length: 0, Exceptions: 0)
Write errors:           0
Non-2xx responses:      2995814
Total transferred:      11452995995 bytes
HTML transferred:       10934720173 bytes
Requests per second:    9985.95 [#/sec] (mean)
Time per request:       10.014 [ms] (mean)
```

```
Time per request:       0.100 [ms] (mean, across all concurrent requests)
Transfer rate:          37281.85 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   5.4      4    1014
Processing:     1    6   3.5      5     229
Waiting:        0    4   2.0      4     207
Total:          2   10   6.5     10    1018

Percentage of the requests served within a certain time (ms)
  50%     10
  66%     11
  75%     11
  80%     12
  90%     13
  95%     15
  98%     18
  99%     20
 100%   1018 (longest request)
```

没啥变化。。

源地址转换

参考：

- https://kubernetes.io/zh/blog/2018/07/09/ipvs-based-in-cluster-load-balancing-deep-dive/
- https://mritd.me/2017/10/10/kube-proxy-use-ipvs-on-kubernetes-1.8/
- https://docs.google.com/document/d/1YEBWR4EWeCEWwxufXzRM0e82l_lYYzIXQiSayGaVQ8M/edit

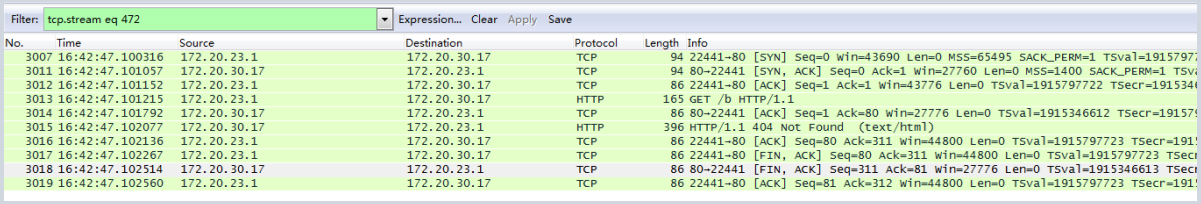
重点说一下 –masquerade-all 选项: kube-proxy ipvs 是基于 NAT 实现的，当创建一个 service 后，kubernetes 会在每个节点上创建一个网卡，同时帮你将 Service IP(VIP) 绑定上，此时相当于每个 Node 都是一个 ds，而其他任何 Node 上的 Pod，甚至是宿主机服务(比如 kube-apiserver 的 6443)都可能成为 rs；按照正常的 lvs nat 模型，所有 rs 应该将 ds 设置成为默认网关，以便数据包在返回时能被 ds 正确修改；在 kubernetes 将 vip 设置到每个 Node 后，默认路由显然不可行，所以要设置 –masquerade-all 选项，以便反向数据包能通过

我在线上是配置的clusterCIDR，没有配置–masquerade-all

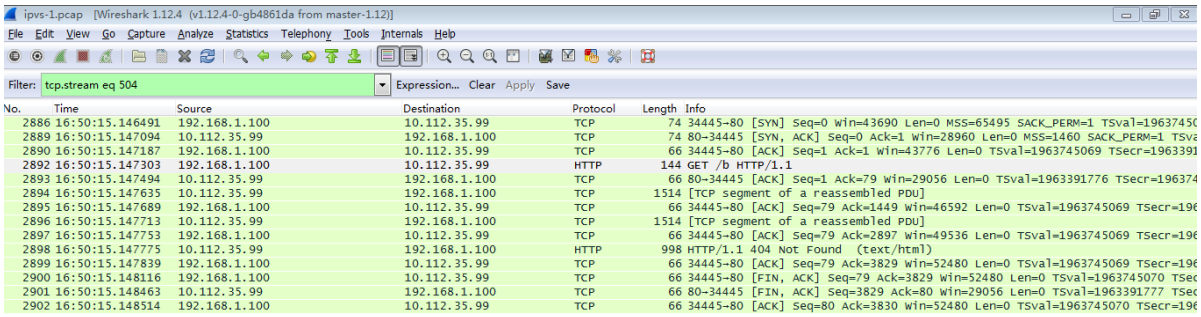
> "clusterCIDR": 必须与 kube-controller-manager 的 --cluster-cidr 选项值一致；kube-proxy 根据 --cluster-cidr 判断集群内部和外部流量，指定 --cluster-cidr<wrap em> 或</wrap> --masquerade-all 选项后 kube-proxy 才会对访问 Service IP 的请求做 SNAT
>
> <
>
> /WRAP>

通过抓包验证，在k8s node上访问clusterIP，原地址被转换成tunl0（calico创建的虚拟隧道）的ip，而自建的源地址就是vip

k8s



独立ipvs



独立ipvs下 手动做snat并删除rs里到vip的路由

```
#iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -j SNAT --to-source 10.112.35.104
```

再次压测

复现k8s中的情况

```
# ab -c100 -t300 -n20000000 -r http://192.168.1.100/b
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.100 (be patient)
apr_pollset_poll: The timeout specified has expired (70007)
Total of 64517 requests completed
```

注意到完成的请求数量和 net.ipv4.ip*local*port_range定义的数量差不太多（宿主机2000到65000，容器是32768到61000，所以容器中压测时只能完成大约3万个请求）

keepalive

如果使用keepalive压测，情况和k8s中也差不多，性能不错

```
# ab -c100 -t300 -n20000000 -r -k http://192.168.1.100/b
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
```

```
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.100 (be patient)
Completed 2000000 requests
Completed 4000000 requests
Completed 6000000 requests
Completed 8000000 requests
Finished 8748252 requests


Server Software:        nginx/1.10.2
Server Hostname:        192.168.1.100
Server Port:            80

Document Path:          /b
Document Length:        3650 bytes

Concurrency Level:      100
Time taken for tests:   300.000 seconds
Complete requests:      8748252
Failed requests:        0
Write errors:           0
Non-2xx responses:      8748301
Keep-Alive requests:    8660823
Total transferred:      33488010519 bytes
HTML transferred:       31931250086 bytes
Requests per second:    29160.81 [#/sec] (mean)
Time per request:       3.429 [ms] (mean)
Time per request:       0.034 [ms] (mean, across all concurrent requests)
Transfer rate:          109010.32 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.9      0    1007
Processing:     0    3  14.9      2     611
Waiting:        0    2   1.8      2     420
Total:          0    3  15.1      2    1011

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      3
  80%      3
  90%      4
  95%      5
  98%      6
  99%      7
 100%   1011 (longest request)
```

k8s中的snat

```
# iptables -vnL -t nat
...
Chain KUBE-MARK-MASQ (3 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK or 0x4000

Chain KUBE-NODE-PORT (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-POSTROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service traffic requiring SNAT */ mark match 0x4000/0x4000
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* Kubernetes endpoints dst ip:port, source ip for solving hairpin purpose */ match-set KUBE-LOOP-BACK dst,dst,src

Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-NODE-PORT  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* Kubernetes nodeport TCP port for masquerade purpose */ match-set KUBE-NODE-PORT-TCP dst
    0     0 KUBE-MARK-MASQ  all  --  *      *      !172.20.0.0/14        0.0.0.0/0            /* Kubernetes service cluster ip + port for masquerade purpose */ match-set KUBE-CLUSTER-IP dst,dst
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match-set KUBE-CLUSTER-IP dst,dst
```

MASQUERADE和SNAT的区别

> 地址伪装，算是snat中的一种特例，可以实现自动化的snat

��iptables中有着和SNAT相近的效果，但也有一些区别，但使用SNAT的时候，出口ip的地址范围可以是一个，也可以是多个，例如：如下命令表示把所有10.8.0.0网段的数据包SNAT成192.168.5.3的ip然后发出去，

```
iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j SNAT --to-source192.168.5.3
```

如下命令表示把所有10.8.0.0网段的数据包SNAT成192.168.5.3/192.168.5.4/192.168.5.5等几个ip然后发出去

```
iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j SNAT --to-source192.168.5.3-192.168.5.5
```

这就是SNAT的使用方法，即可以NAT成一个地址，也可以NAT成多个地址，但是，对于SNAT，不管是几个地址，必须明确的指定要SNAT的ip，假如当前系统用的是ADSL动态拨号方式，那么每次拨号，出

口ip192.168.5.3都会改变，而且改变的幅度很大，不一定是192.168.5.3到192.168.5.5范围内的地址，这个时候如果按照现在的方式来配置iptables就会出现问题了，因为每次拨号后，服务器地址都会变化，而iptables规则内的ip是不会随着自动变化的，每次地址变化后都必须手工修改一次iptables，把规则里边的固定ip改成新的ip，这样是非常不好用的。

MASQUERADE就是针对这种场景而设计的，他的作用是，从服务器的网卡上，自动获取当前ip地址来做NAT。 比如下边的命令：


```
iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j MASQUERADE
```

如此配置的话，不用指定SNAT的目标ip了，不管现在eth0的出口获得了怎样的动态ip，MASQUERADE会自动读取eth0现在的ip地址然后做SNAT出去，这样就实现了很好的动态SNAT地址转换。

作者：siaisjack 来源：CSDN 原文：https://blog.csdn.net/jk110333/article/details/8229828版权声明：本文为博主原创文章，转载请附上博文链接！

调整内核参数

- https://jsravn.com/2017/12/24/ipvs-with-kubernetes-ingress/
- https://github.com/cloudnativelabs/kube-router/issues/544
- https://github.com/moby/moby/issues/35082
- https://github.com/moby/moby/issues/35082#issuecomment-419072026

- conn*reuse*mode default 1. 改为0，完成大约300万请求，性能正常

  问题解决：
  https://github.com/kubernetes/kubernetes/issues/70747#issuecomment-437558941

副作用

- https://github.com/cloudnativelabs/kube-router/issues/544#issuecomment-431553463

> Our tests showed that disabling reuse with 'net.ipv4.vs.conn_reuse_mode=0' will interfere with scaling. When adding more pods in a high traffic scenario the traffic will stick to the old and overloaded pods and when scaling down, the traffic will be send to non existent pods.

下图中时间是UTC时间

事件

消息	来源	子对象	总数	最早出现于	最近出现于
New size: 17; reason: All metrics below target	horizontal-pod-autoscaler	-	7	2018-11-03T13:46 UTC	2018-11-10T14:24 UTC
Scaled down replica set device-manager-api-internal-5486944c9b to 17	deployment-controller	-	6	2018-11-05T12:22 UTC	2018-11-10T14:24 UTC
New size: 19; reason: All metrics below target	horizontal-pod-autoscaler	-	3	2018-11-07T13:37 UTC	2018-11-10T14:05 UTC
Scaled down replica set device-manager-api-internal-5486944c9b to 19	deployment-controller	-	3	2018-11-07T13:37 UTC	2018-11-10T14:05 UTC
New size: 21; reason: All metrics below target	horizontal-pod-autoscaler	-	1	2018-11-10T13:43 UTC	2018-11-10T13:43 UTC
Scaled down replica set device-manager-api-internal-5486944c9b to 21	deployment-controller	-	1	2018-11-10T13:43 UTC	2018-11-10T13:43 UTC

下图中时间是UTF+8

可能的解决方案

expire*nodest*conn没有效果

/proc/sys/net/ipv4/vs/expire*nodest*conn

默认值为0，当LVS转发数据包，发现目的RS无效（删除）时，会丢弃该数据包，但不删除相应 连接；这样设计的考虑是，RS恢复时，如果Client和RS socket还没有超时，则 可以继续通讯；如果将该参数置1，则马上释放相应 连接；

expire_nodest_conn - BOOLEAN
0 - disabled (default)
not 0 - enabled

The default value is 0, the load balancer will silently drop packets when its destination server is not available. It may be useful, when user-space monitoring program deletes the destination server (because of server overload or wrong detection) and add back the server later, and the connections to the server can continue.

If this feature is enabled, the load balancer will expire the connection immediately when a packet arrives and its destination server is not available, then the client program

will be notified that the connection is closed. This is equivalent to the feature some people requires to flush connections when its destination is not available.

Printed on: 2022/10/05 18:01

Convert to img Failed!