Kubernetes测试报告

Wen Zhenglin, Zhai Xigui
2018-5-10

目录

1 测试总结

通过对可用性测试，验证了Kubernetes在出现节点宕机和各组件出现故障时对业务的影响，通过压力测试得出了一些性能指标及压力测试时是否产生其它重大问题。

测试结果未发现Kubernetes的重大问题。

可用性测试发现Kubernetes仍然比较稳定，并且没有出现故障不能恢复的情况。

网络压力测试发现Kubernetes节点间的网络性能相对较差，通过讨论得出属于正常范围。

Pods压力测试发现Dashboard有出现页面不能响应问题，和没有具体影响的ErrImagePull事件，不是重大问题。

Docker和虚拟机的服务性能对比发现性能基本接近，无显著区别。

同时发现多实例没有带来性能的提升，但能处理更大的并发请求量。

通过此次测试，得到了一些性能指标和以下问题：

2 可用性测试

2.1 系统崩溃测试

通过节点Crash的方式，测试节点重启或组件重启，能不能自动恢复和是否对业务造成影响。

2.1.1 测试方法

1. 模拟Crash
2. 通过Prometheus+Cloudprobe验证服务的状态

Crash命令：`echo c > /proc/sysrq-trigger`

Nginx已有相应自动切换配置，如发现错误返回码将在其它节点重试该请求（目前只在10.66.8.66上有此配置），用于验证Crash无影响。

2.1.2 测试结果

测试描述	历时	结果	备注
K8s node crash -10	2m	Service ok	
K8s node crash -11	2m	Service ok	
K8s node crash -12	2m	Service ok	
K8s node crash -18	2m	Service ok	
Etcd leader crash (170)	3m15s	Service ok	
Etcd node crash (171)	手动启动	Service ok	
Etcd leader crash (172)	手动启动	Service ok	
K8s master crash (169)	手动启动	Service ok, kubectl cli few second down	
K8s master crash(168)	手动启动	Service ok	Virutal ip down (config error)
K8s master crash (169) again	手动启动	Service ok	Keepalived not up
Etcd two nodes crash	20minutes	Service ok, Kubectl down	One of nginx not configure upstream
Two master and two etcd crash	30 minutes	Service ok, Kubectl down	Also

			stopped 18's kubelet
K8s kube-scheduller stop	手动启动	Service ok	
K8s kube-controller-manager stop	15s	Service ok	Auto switch over
K8s kube-apiserver stop	18s	Service ok	Auto switch over

注：

1. 10,11,12,18代表节点ip后辍，对应10.66.0.10,10.66.0.11,10.66.0.12,10.66.0.18
2. 10.66.8.170,171,172 (etcd nodes), 10.66.8.168,169 (master nodes)
3. Service ok 代表Cloudprobe每秒监控ok (每秒定期访问nginx接四个fs后端)
4. VM需要手动启动
5. Service stop 通过systemctl stop

2.1.3 问题1：Kube-controller-manager无故退出

此问题目前对业务无影响，暂未发现导致的原因

```
Apr 21 17:23:40 bjsh66-8-168.qbos.com kube-controller-manager[1523]: I0421
17:23:40.081940 1523 leaderelection.go:208] failed to renew lease kube-s
ystem/kube-controller-manager
Apr 21 17:23:40 bjsh66-8-168.qbos.com kube-controller-manager[1523]: F0421
17:23:40.082170 1523 controllermanager.go:216] leaderelection lost
Apr 21 17:23:40 bjsh66-8-168.qbos.com systemd[1]:
kube-controller-manager.service: main process exited, code=exited,
status=255/n/a
```

2.1.4 问题2： 虚拟机Crash需要手工重启

Crash方式：`echo c > /proc/sysrq-trigger`

此问题暂不能解决？目前重启对业务可能造成影响的节点都是实际物理机，也一定程度避免了影响，目前的测试结果虚拟机上跑的服务（Master,Etcd)重启未发现对正在运行的业务造成影响。

2.1.5 问题3：Journalctl无可用的数据

有可能是Crash导致的，目前只遇到了一次，开户Journalctl的Debug日志，找到失败的条目，移除该文件即可，可能造成日志缺少一部分。

```
[root@bjsh66-8-169 system]# journalctl
Error was encountered while opening journal files: 没有可用的数据

# SYSTEMD_LOG_LEVEL=debug journalctl
...
Failed to add file
/var/log/journal/4d2b7e1a19414cd886943094a4a1734f/system@00056a7e659fcbf0-b
fc9c244ebeb0f54.journal~: 没有可用的数据
...
```

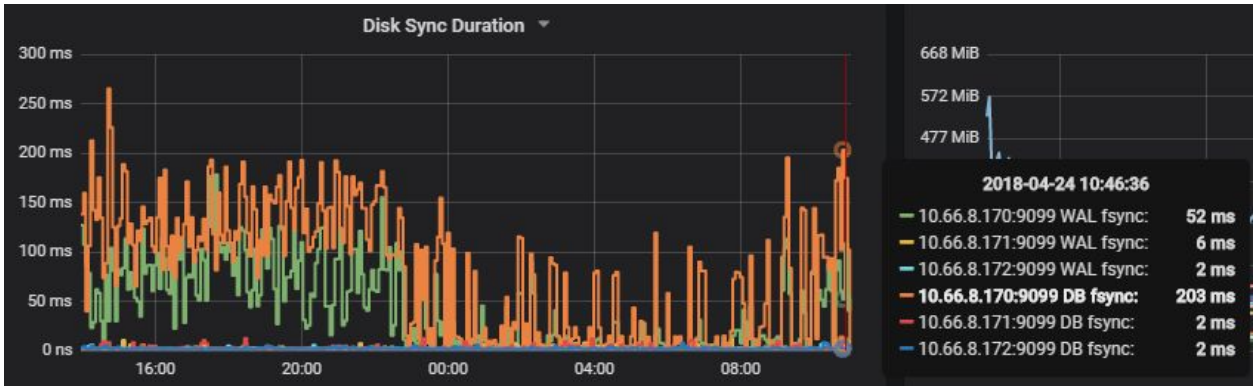
移除该文件即可。

2.1.6 问题4：Etcd一个节点170磁盘写同步较高

10.66.8.170上的DB fsync指标普遍维持在100ms以上（另外两个节点维护在10ms以内），Leader切到别的节点稍微有点缓解，目前仍在100ms-200ms左右。

同时etcd打印异常日志：

```
4月 24 17:30:30 bjsh66-8-170.qbos.com etcd[970]: failed to send out
heartbeat on time (exceeded the 100ms timeout for 136.935934ms)
4月 24 17:30:30 bjsh66-8-170.qbos.com etcd[970]: server is likely
overloaded
...
4月 24 17:32:15 bjsh66-8-170.qbos.com etcd[970]: apply entries took too
long [101.132328ms for 1 entries]
4月 24 17:32:15 bjsh66-8-170.qbos.com etcd[970]: avoid queries with large
range/delete range!
```



目前未发现影响，有待进一步观察。

2.1.7 问题5：硬件时间导致节点Not ready

通过18上的Crash测试，发现节点长时间Not ready，经过排查发现，为硬件时间问题导致。

```
[root@bjsh66-0-18 ~]# kubectl get nodes
NAME          STATUS     AGE       VERSION
10.66.0.11    Ready      125d      v1.6.8
10.66.0.12    Ready      131d      v1.6.8
10.66.0.18    NotReady   109d      v1.6.8
[root@bjsh66-0-18 ~]#
```

经过查看日志发现，节点上的时间存在偏差，系统启动后，恢复了正常时间（服务启动一开始打印的日志时间为错的，大约一分钟Systemd调整了时间），但节点Kubelet与Kube-Controller-Manager的发送的心跳时间仍是错误的时间，导致k8s不认为节点已经恢复。

```
4月 04 15:30:04 bjsh66-0-10.qbos.com kube-controller-manager[13555]: I0404
15:30:04.781711   13555 nodecontroller.go:1007] node 10.66.0.18 hasn't been
updated for 15m20.204801669s. Last ready condition is: {Type:Ready
Status:Unknown LastHeartbeatTime:2018-04-04 15:49:00 +0800 CST
LastTransitionTime:2018-04-04 15:15:24.583762723 +0800 CST
Reason:NodeStatusUnknown Message:Kubelet stopped posting node status.}
  //日志时间变化
4月 04 15:48:55 bjsh66-0-10.qbos.com kube-controller-manager[13555]: I0404
15:48:55.015944   13555 nodecontroller.go:1035] node 10.66.0.18 hasn't been
updated for 34m10.439036616s. Last DiskP
ressure is:
&NodeCondition{Type:DiskPressure,Status:Unknown,LastHeartbeatTime:2018-04-0
4 15:49:00 +0800 CST,LastTransitionTime:2018-04-04 15:15:24.583763014 +0800
```

```
CST,Reason:NodeStatusUnknown
,Message:Kubelet stopped posting node status.,}
```

```
//这时系统时间与错误时间相一致，节点恢复

4月 04 15:49:00 bjsh66-0-10.qbos.com kube-controller-manager[13555]: I0404
15:49:00.016339   13555 nodecontroller.go:973] ReadyCondition for Node
10.66.0.18 transitioned from Unknown to &Node
Condition{Type:Ready,Status:True,LastHeartbeatTime:2018-04-04 15:48:58
+0800 CST,LastTransitionTime:2018-04-04 15:48:58 +0800
CST,Reason:KubeletReady,Message:kubelet is posting ready status,}
```

经过多次复现和验证，确认是硬件时间导致的Node not ready

目前只发现18上有硬件时间问题，其余节点正常

解决办法如下：

```
[root@bjsh66-0-18 ~]# hwclock --show
2018年04月04日 星期三 18时59分12秒  -0.156798 秒
[root@bjsh66-0-18 ~]# hwclock -w
[root@bjsh66-0-18 ~]# hwclock --show
2018年04月04日 星期三 18时25分01秒  -0.250575 秒
[root@bjsh66-0-18 ~]#
```

调整硬件时间后，再次Crash验证，确认节点重启后15分钟内未再出现Node not ready状态。

3 压力测试

3.1 网络测试

通过网络测试查看网络的性能指标，网络丢包，以及相关的潜在问题。

3.1.1 测试方法

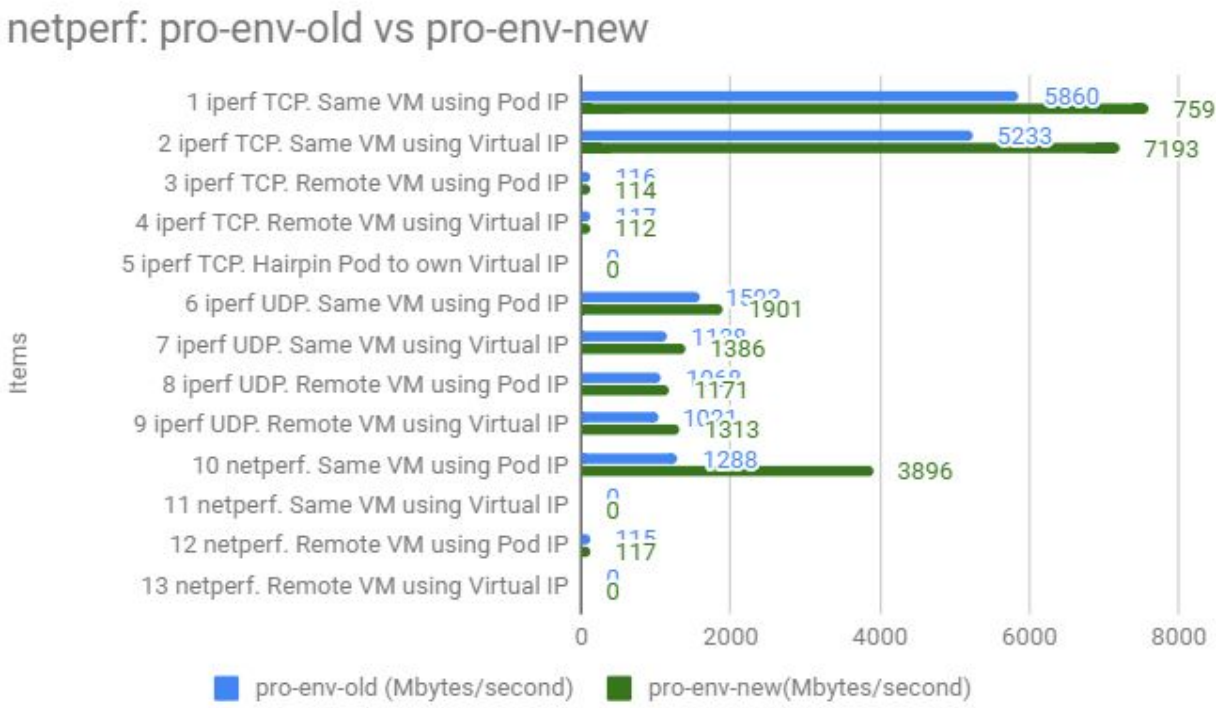
https://github.com/kubernetes/perf-tests/tree/master/network/benchmarks/netperf

1. compile netperf
2. docker pull sirot/netperf-latest
3. ./netperf -kubeConfig ~/.kube/config

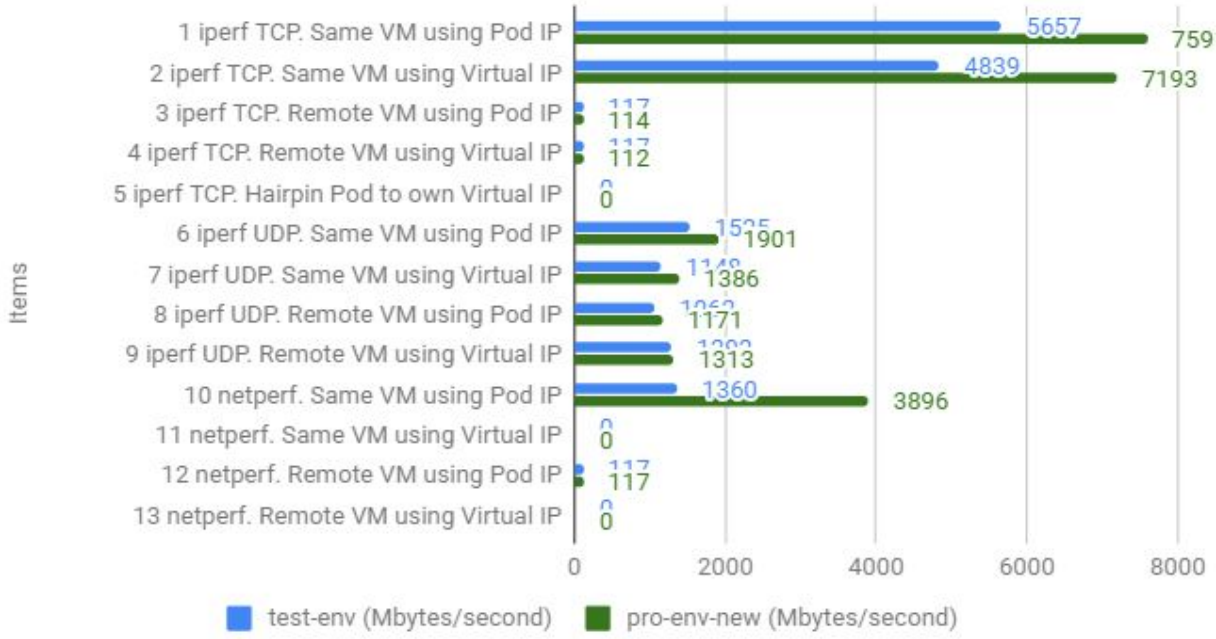
3.1.2 测试结果

根据目前的测试结果，各项指标基本正常，有少量丢包现象，以及已知问题导致单播泛洪现象。

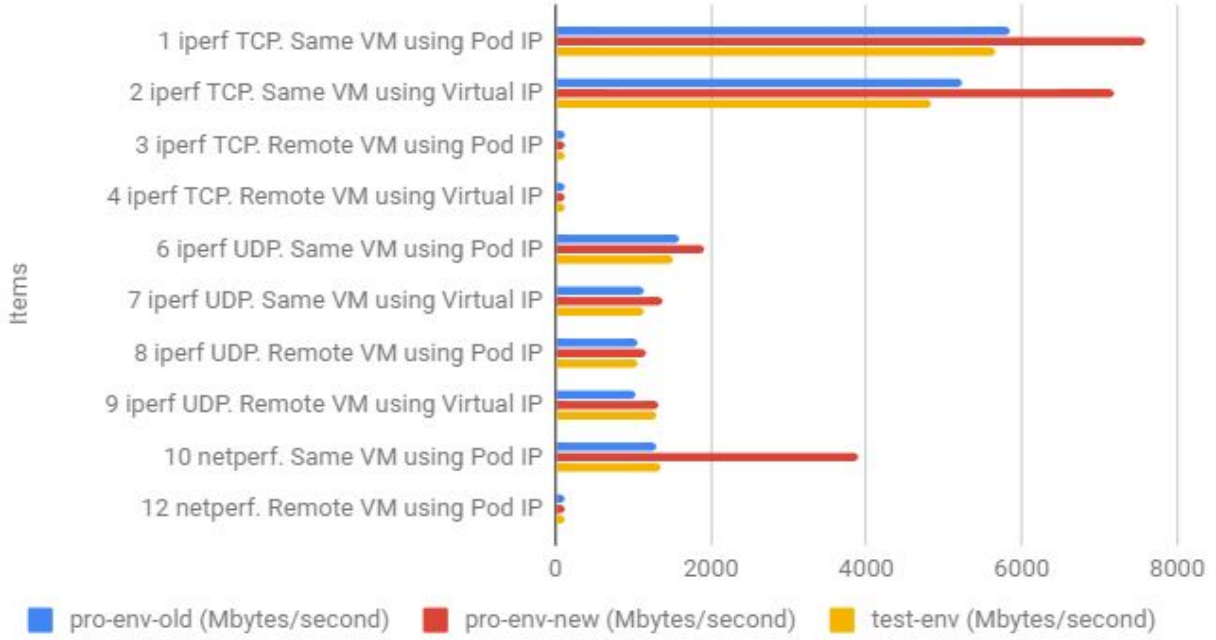
并进行了多环境的对比，发现新搭建的集群版本有比较大的提升。



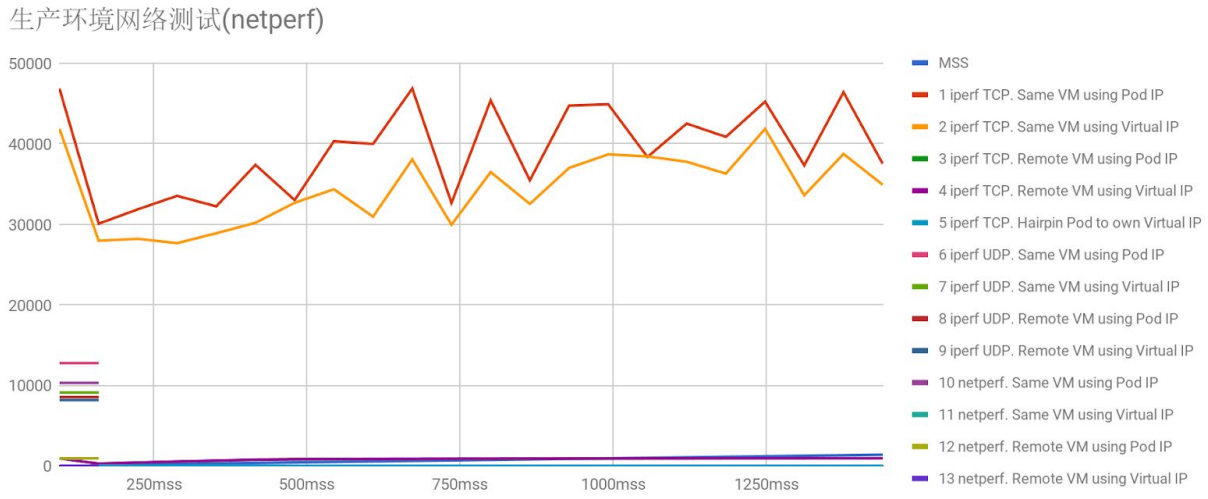
netperf: test-env vs pro-env-new

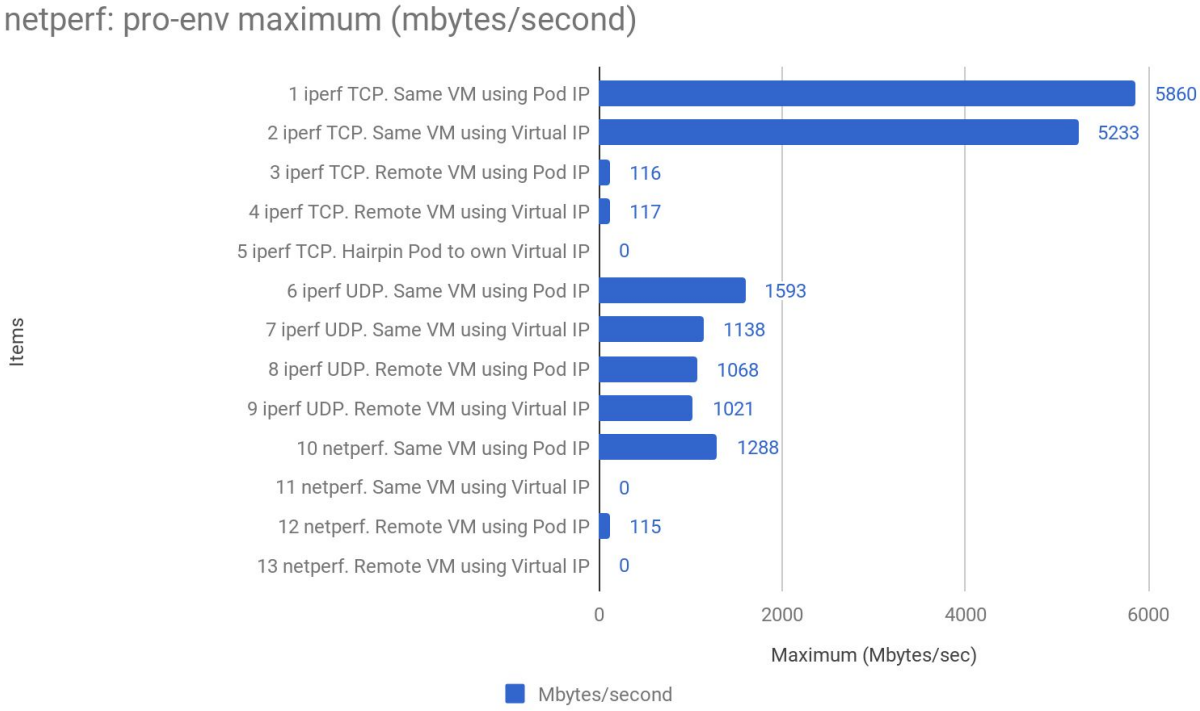


netperf: pro-env-old, pro-env-new and test-env

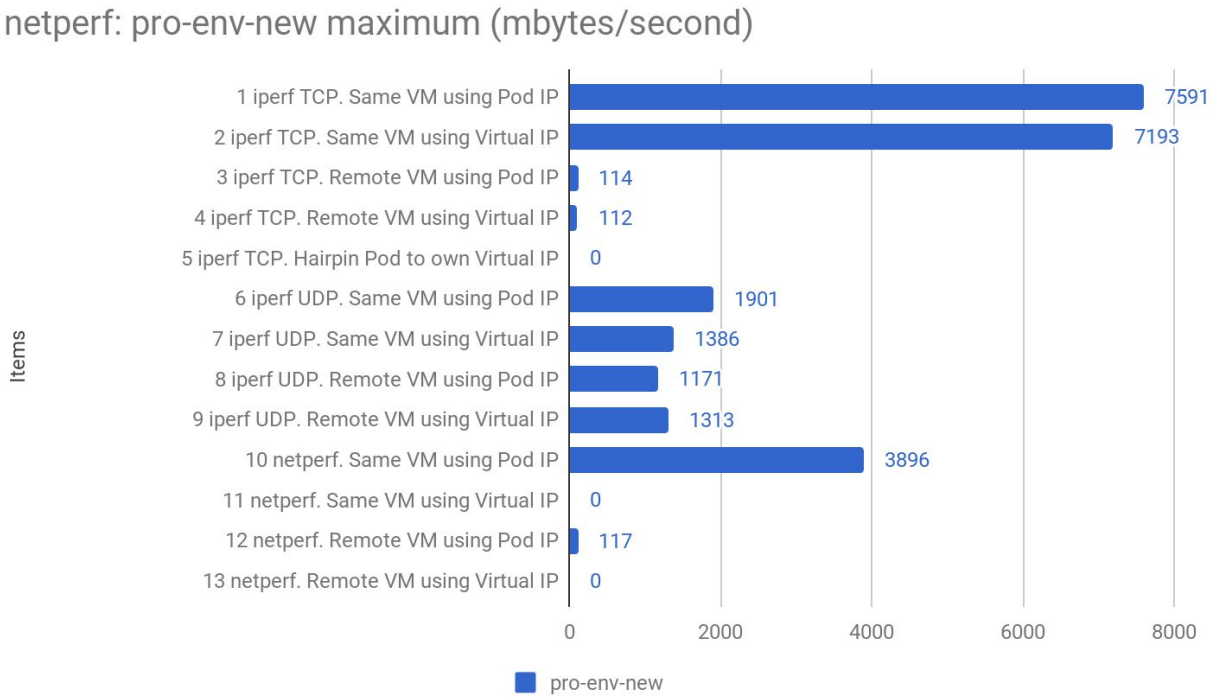


3.1.3 生产环境



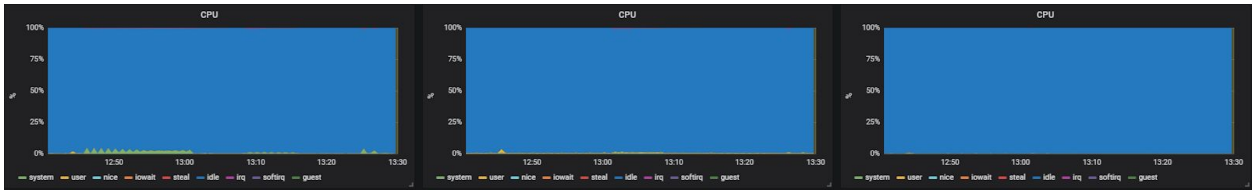


由于在原有的集群 上升级了小版本，从v1.6.8至v1.6.13，下面为新集群的网络测试结果：



以下为新集群的相关数据

测试前CPU基本为空闲的状态：



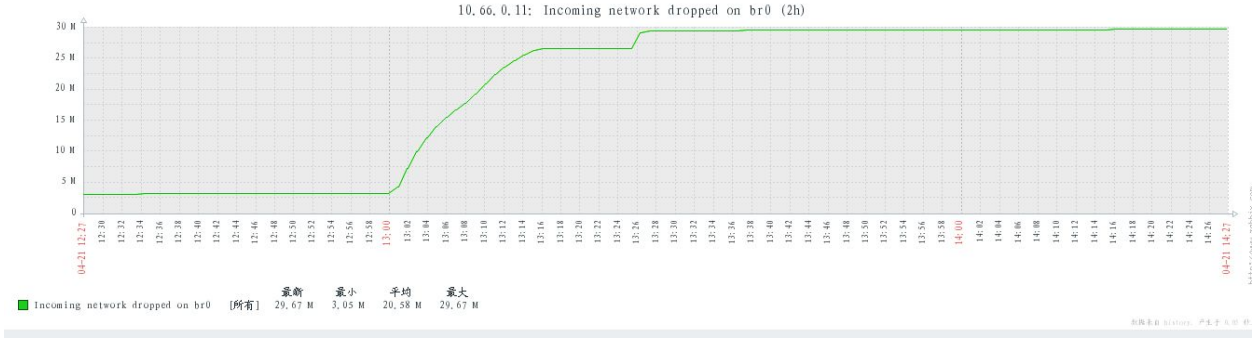
丢包相关数据：

服务器	网络流量 (M)	丢包(M)
10.66.0.11	476	29
10.66.0.10	484	36
10.66.0.18	490	273

目前丢包的原因预计和Kubernetes的网络插件和我们的网络环境有关，导致有单播泛洪的现象。

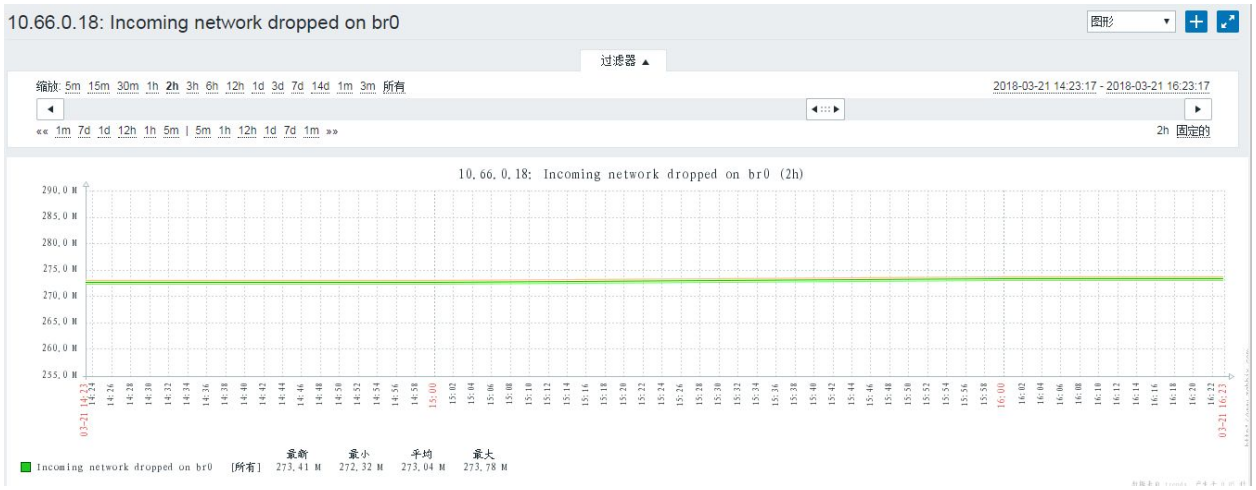
下面为具体的Zabbix相关网络监控数据 ：

10.66.0.11: Incoming network dropped on br0



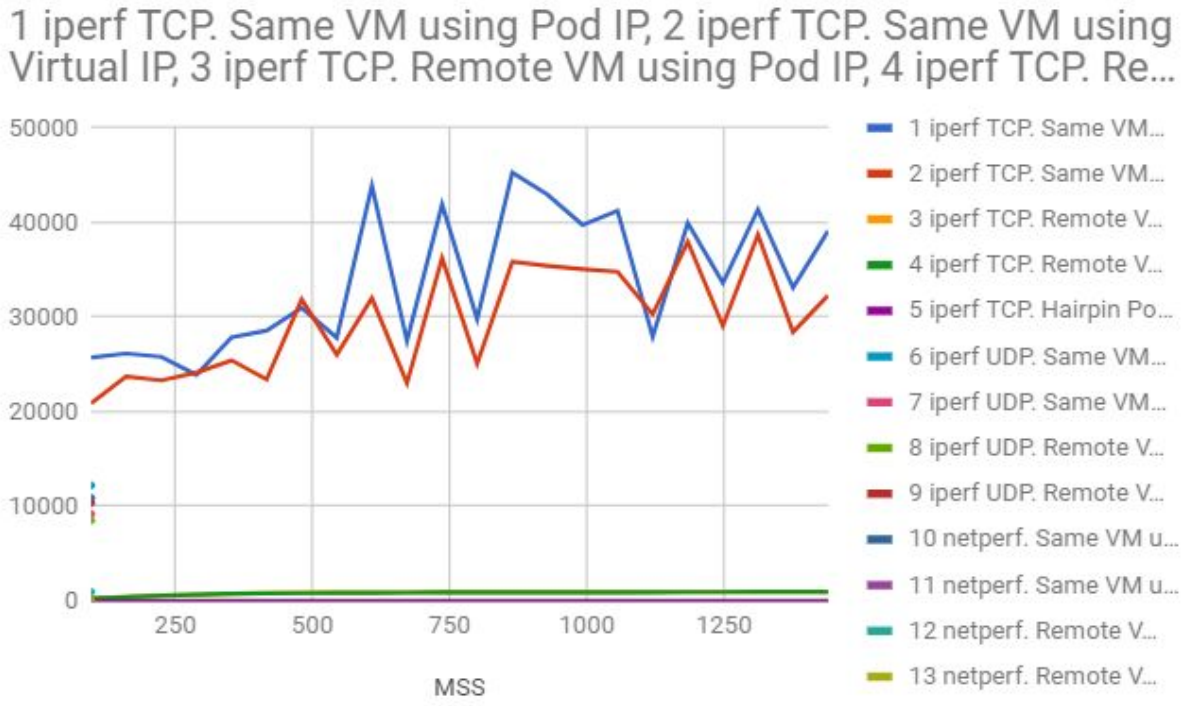
10.66.0.10: Incoming network dropped on br0

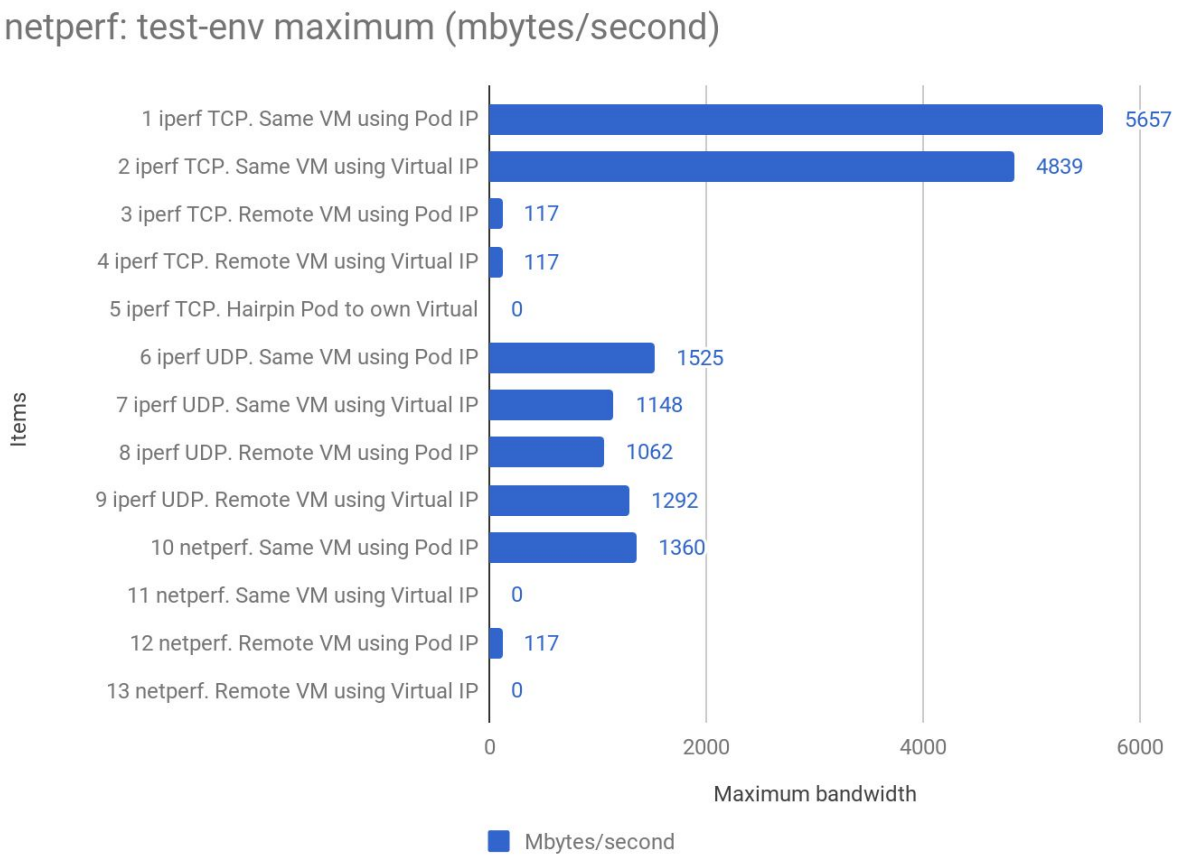
10.66.0.18: Incoming network dropped on br0



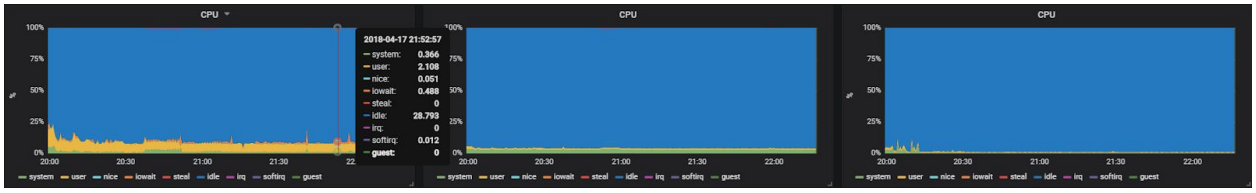
3.1.4 测试环境

通过和测试环境的测试发现，生产环境和测试环境网络性能基本接近。

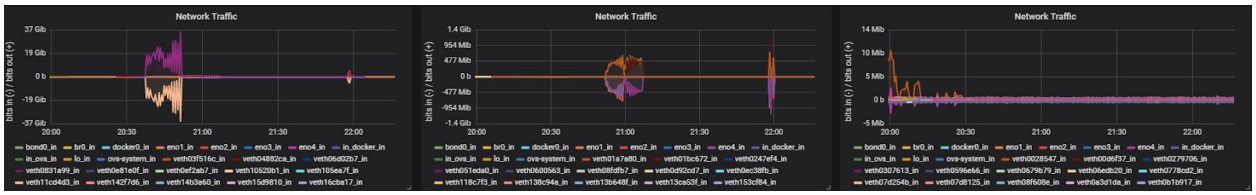




由于Kubernetes Master在其中一个节点上（123），该节点cpu有少量使用。



网络流量：



3.1.5 问题1：单播泛洪（已知问题）

此问题为已知问题，出现的丢包指标是由于单播泛洪导致，目前没有很好的解决方案，经过同事验证，只在Docker上才有，目前怀疑和我们的网络插件方案Open VSwitch有关。

3.2 Pods压力测试

3.2.1 测试方法

测试项：

- Test1：单节点指定资源最大Pod数
- Test2：单节点不指定资源最大Pod数
- Test3：所有节点指定资源最Pod数
- Test4：所有节点不指定资源最大Pod数
- Test5：Pod创建和销毁测试
- Test6：Deployment创建和销毁测试
- Test7：长时间大规模创建和删除测试

参考：
https://docs.openstack.org/developer/performance-docs/test_plans/kubernetes_proxy/plan.html
https://docs.openstack.org/developer/performance-docs/test_results/container_cluster_systems/kubernetes/pod_recovery/index.html#kubernetes-pod-recovery-test-report
https://github.com/AleksandrNull/MMM

我们基于OpenStack做的Pods相关测试的基础上，做了少许改动。创建了以上7个测试项的测试，具体见：

http://issue.qianbao-inc.com/SRE/k8s/src/branch/master/testing/pod-tests/README.md

测试前需要在每个节点提前下载镜像

docker pull reg.qianbao-inc.com/k8s/mmm-master:latest
docker pull reg.qianbao-inc.com/k8s/mmm-minion:latest
docker pull reg.qianbao-inc.com/k8s/mmm-mariadb:latest

测试时资源限制的设置为1G内存，CPU不指定

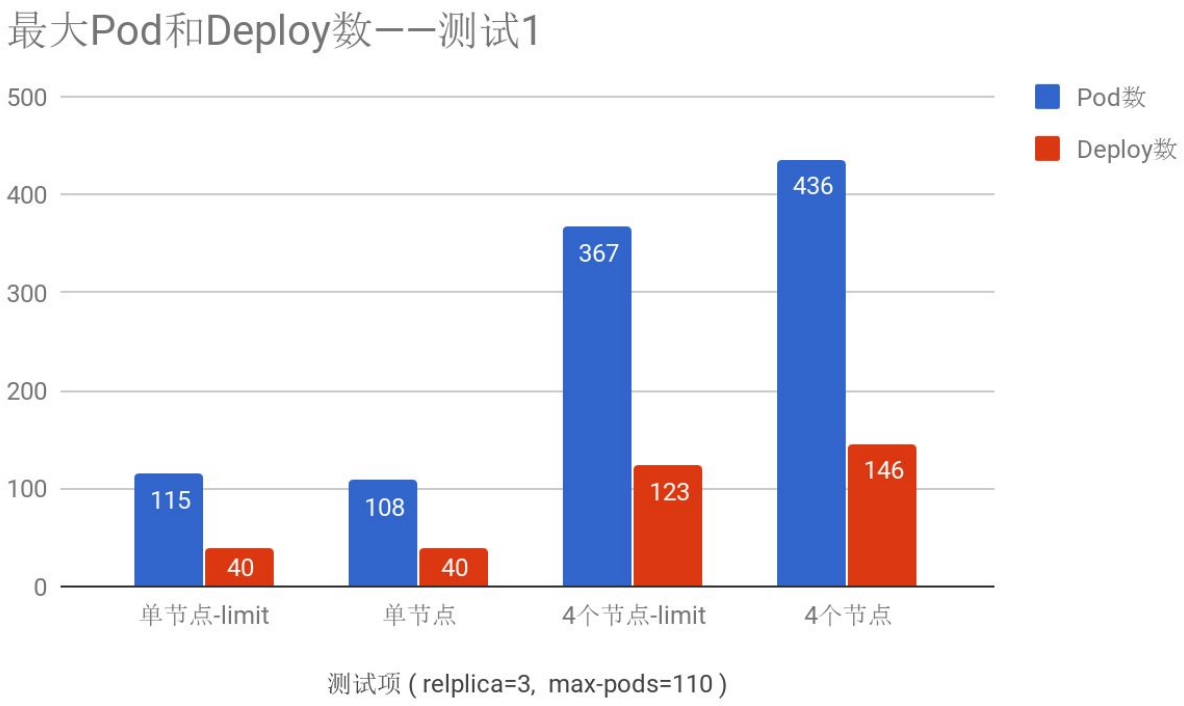
3.2.2 测试结果

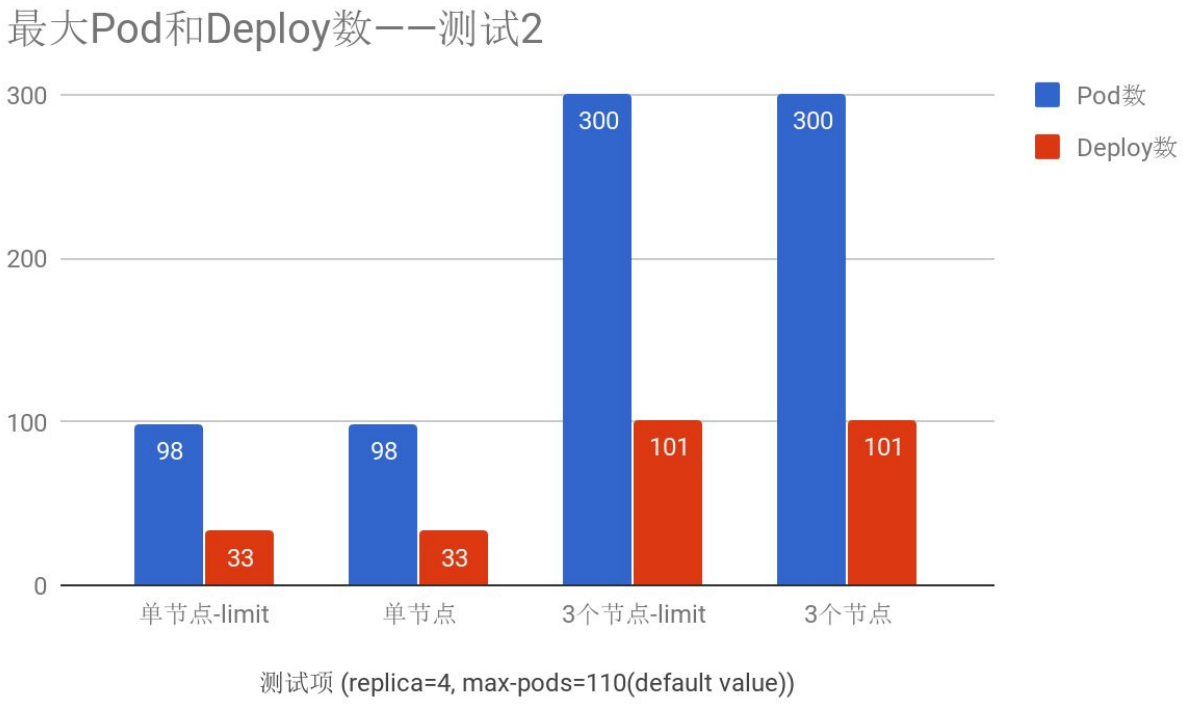
通过此次测试能知道大致的Pods上限，同时发现Kubelet参数max-pods将限制Pods创建，我们在所有Kubelet节点上修改此限制至250，并进行了测试，考虑到ip资源的问题（目前只有C段的一半，即128个ip可分配），目前已仍以max-pods默认值110测试为主。

通过此次测试未发现重大问题，目前遇到的有：

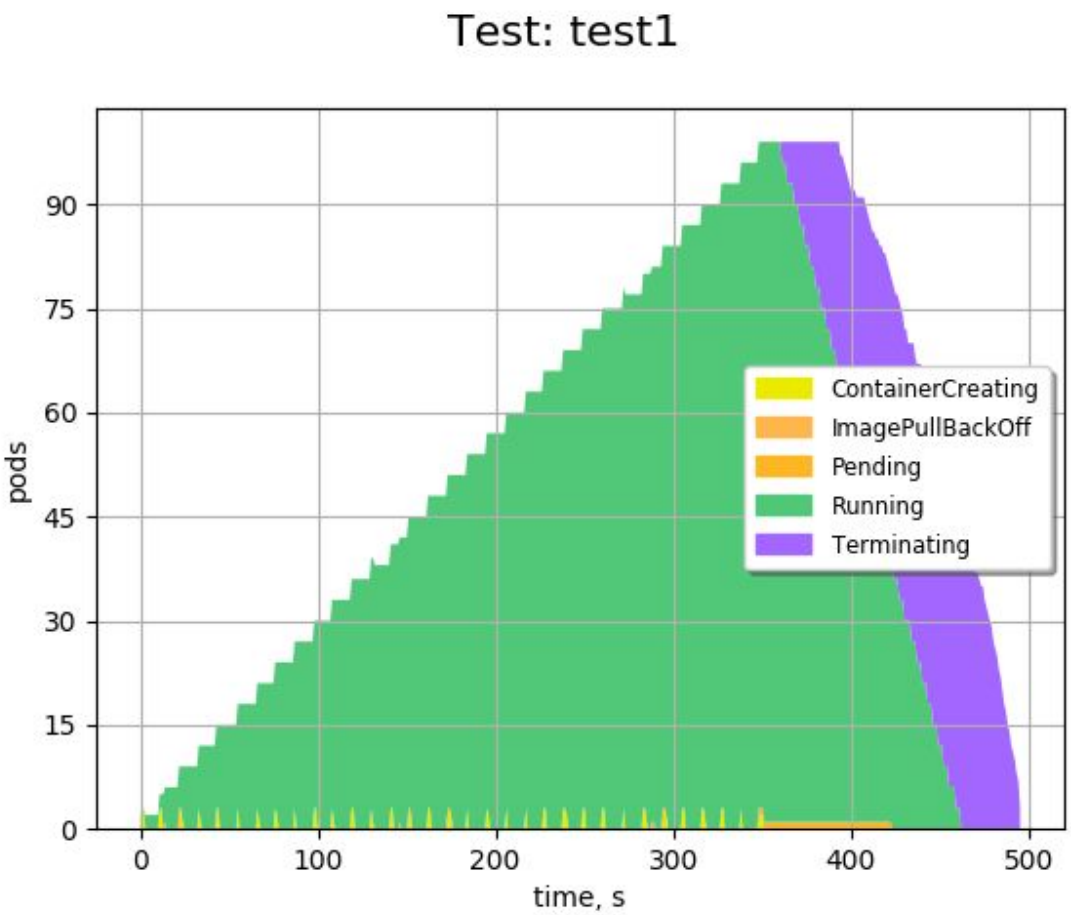
1. 出现一定频率的镜像失败事件（ErrImagePull），发现并无具体影响，Pod成功创建。
2. Dashboard在一个名字空间里查看Pods数大于140左右将出现错误，页面不能显示。

测试结果如下：





下面为max-pods为110时的各项的测试的结果：

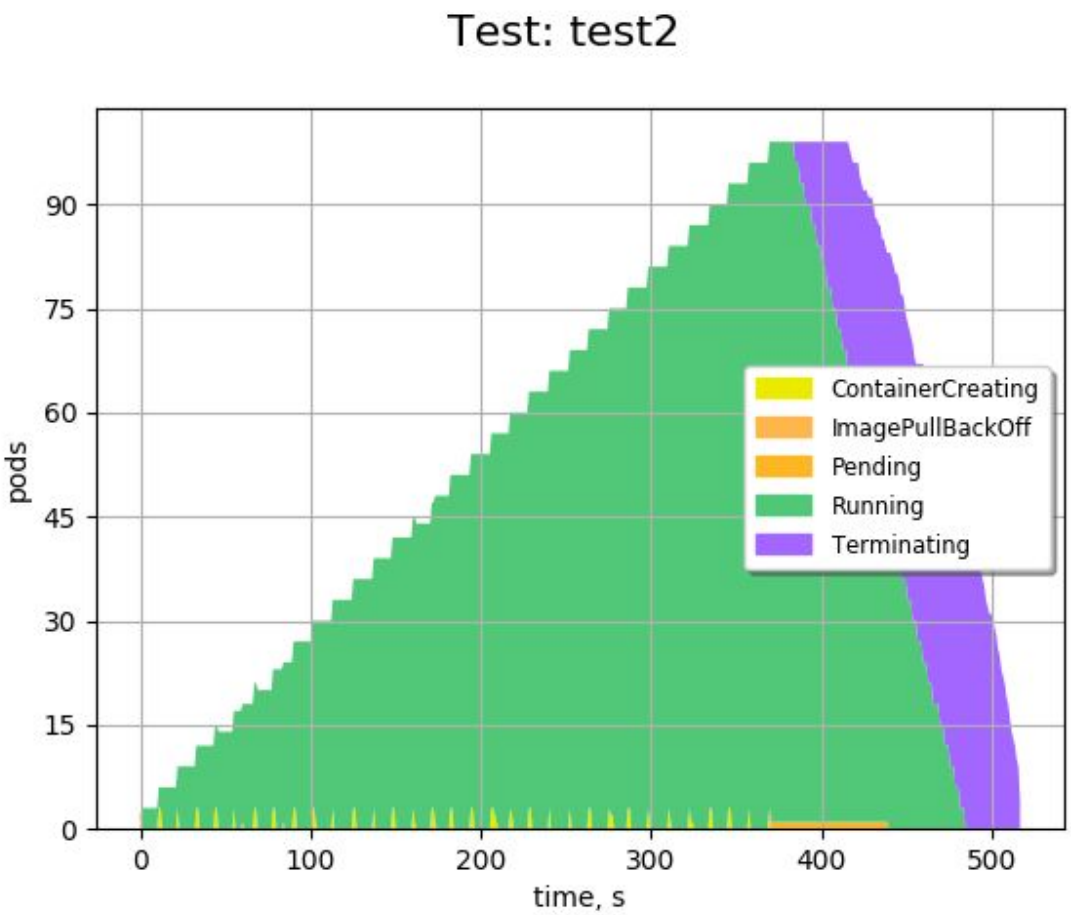



```
2s        2s        2         minion-44-333091213-clhdf    Pod
Warning   FailedScheduling    default-scheduler        No nodes are
available that match all of the following predicates:: Insufficient memory
(1), MatchNodeSelector (3).
```

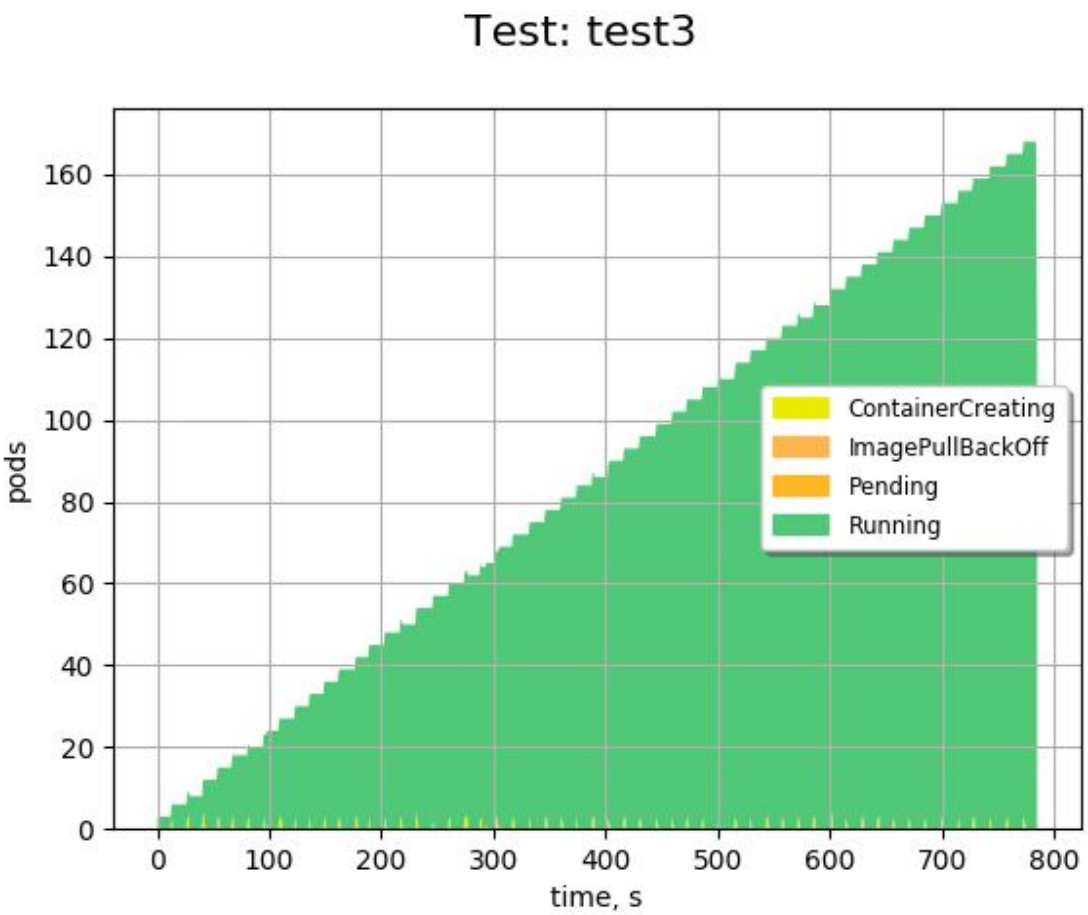
由于生产环境内存及CPU资源较大，目前主要的限制条件为max-pods，在其它测试如测试环境中有遇到CP、内存和ip地址等资源不足的情况。

上图属正常现象，所有的Pods支行后，正常销毁了，创建Pods的时间平均在4秒。
少量的Imagepull backoff状态未发现具体影响。

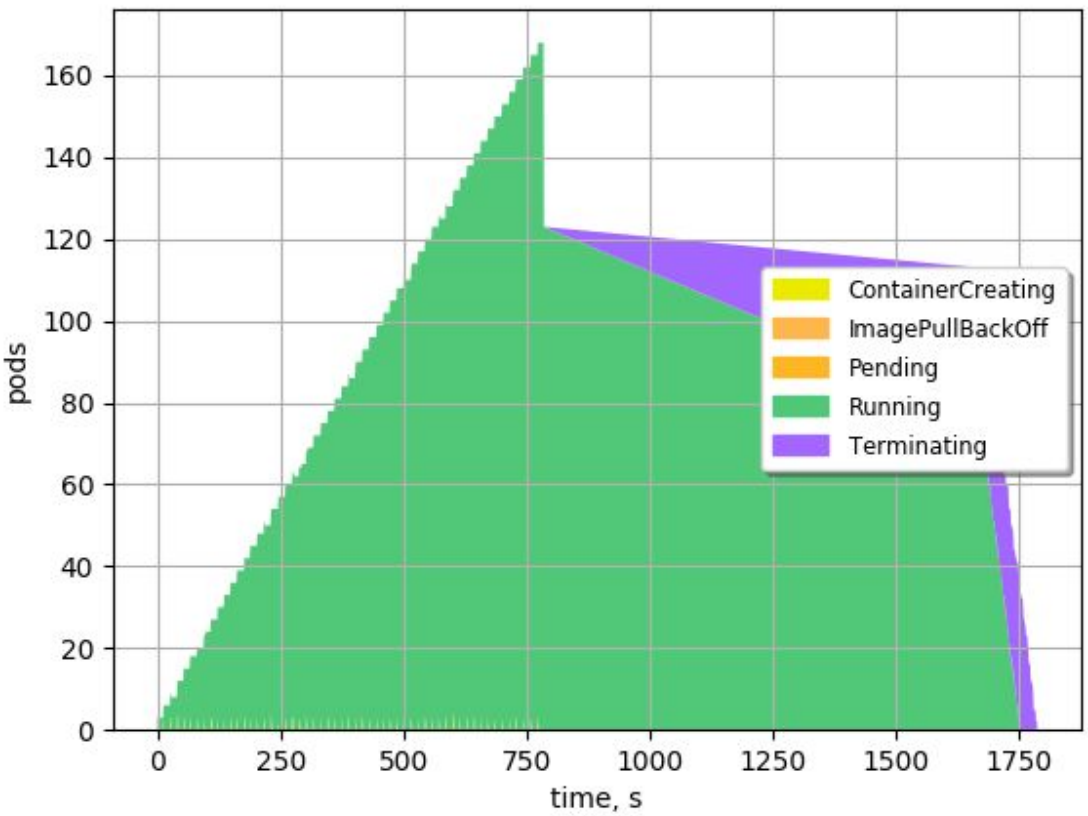
测试目前为每个Pod设的资源为1G内存。

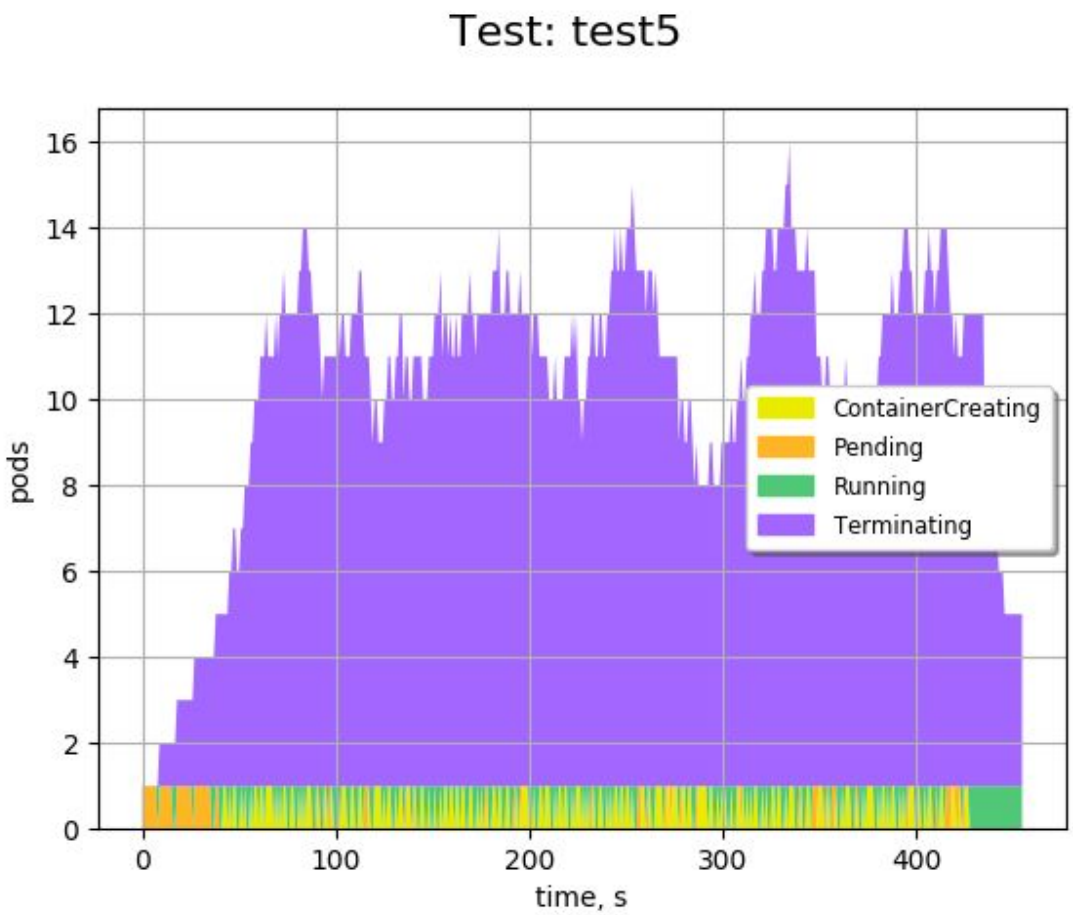


由于max-pods的限制，所以Test2和Test1基本无区别（业务使用时，具体的Pods数可能由于资源的限制预计为更少的Pods数）。

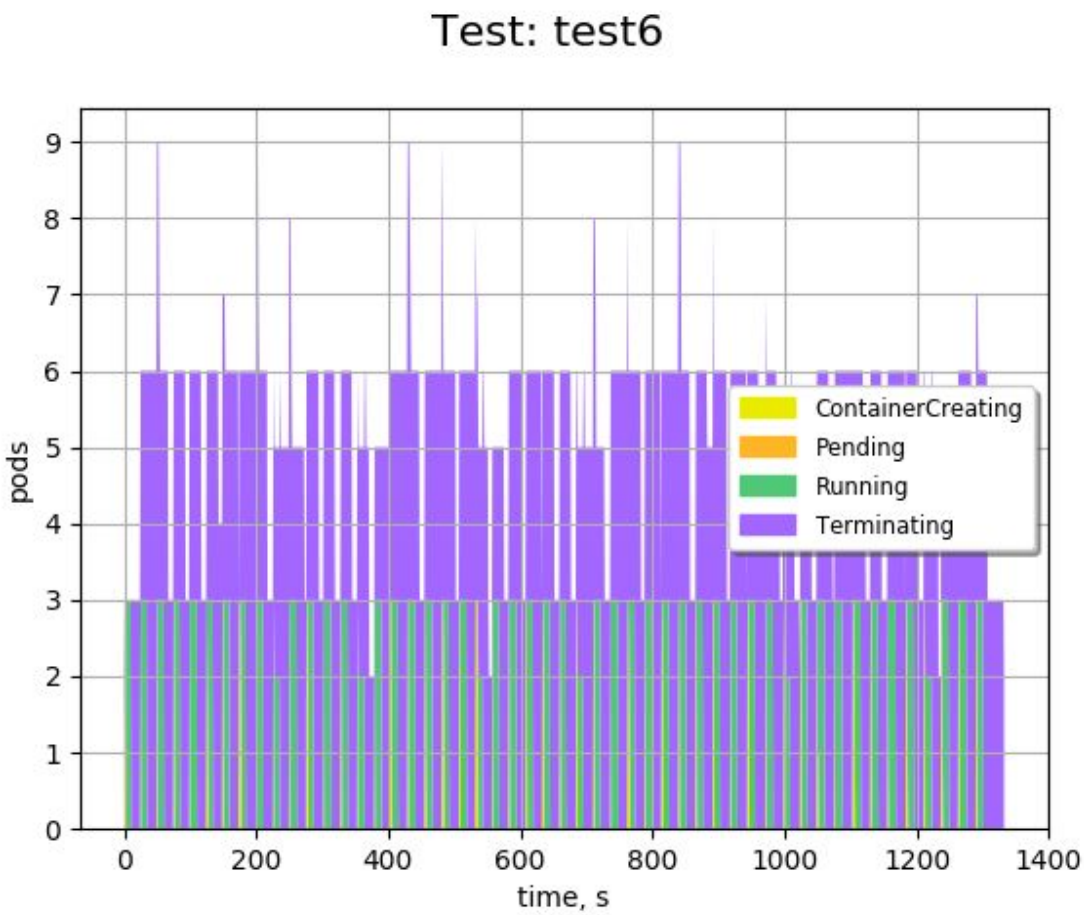


注：测试过程中，我们的的数据收集仍存在一些问题，数据采集中间停止了，测试结束都是正常删除的，无发现问题（排查过程中通过手工启动，并叠加数据，通过下图可以看出是正常删除了）

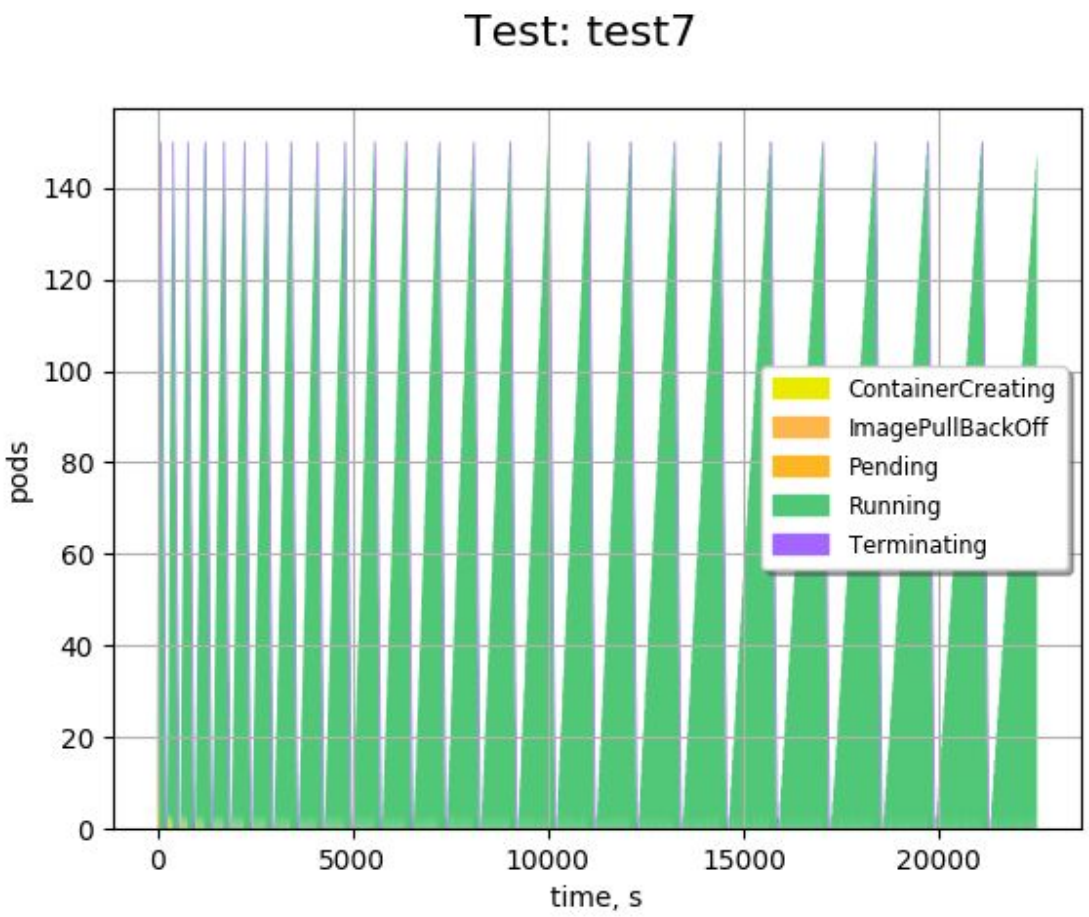

Test: test3_append
pods
time, s
0
20
40
60
80
100
120
140
160
0
250
500
750
1000
1250
1500
1750
ContainerCreating
ImagePullBackOff
Pending
Running
Terminating



Test5主要是通过多次创建和销毁Pods来验证是否发现Kubernetes的相关问题，目前比较稳定。



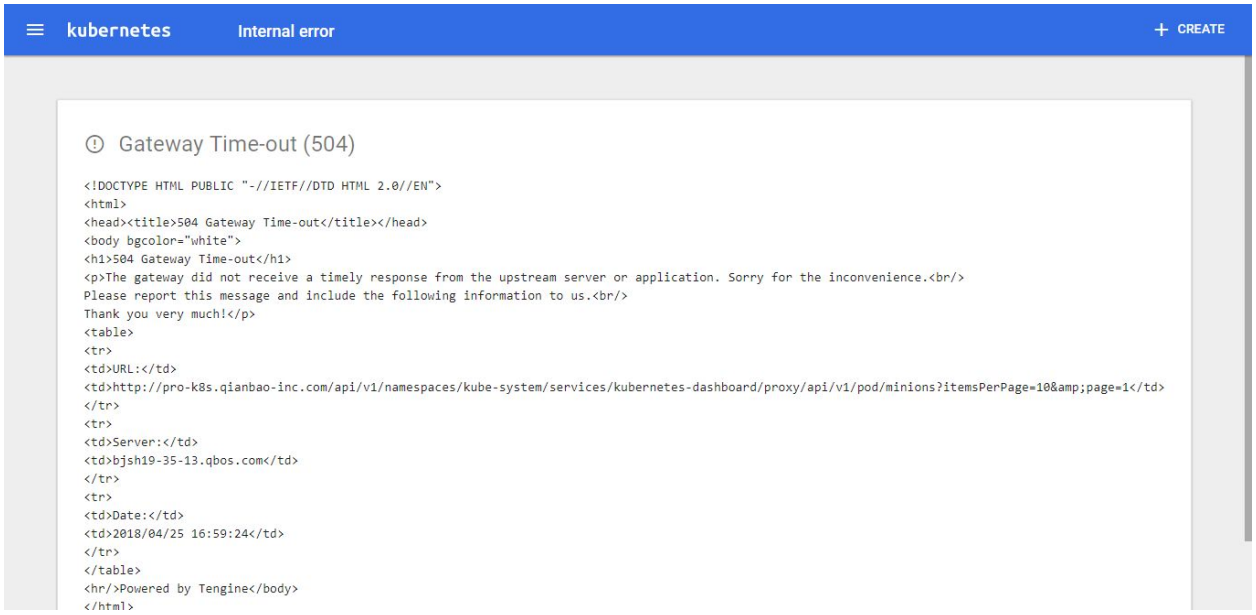
Test6主要是通过多次创建和销毁Deployment来验证是否发现Kubernetes的相关问题，目前比较稳定。



Test7主要是通过大规模创建和销毁Deployment来验证是否发现Kubernetes的相关问题，一次创建 最大数的Deployment，然后删除，如此反复，执行超过5.5小时（50次），目前比较稳定，未发现问题。

3.2.3 问题1：Dashboard查看Pods页面返回504

在测试过程中发现Dashboard查看Pods时出现504，页面不能正常响应，命令行查看是正常的。因此判定是正常的，同时也尝试了一下是否有潜在的其它原因导致的，目前发未发现其它问题。



出现此问题的现象在于Pods数，当名字空间内Pods数在大于140左右会出现此问题。

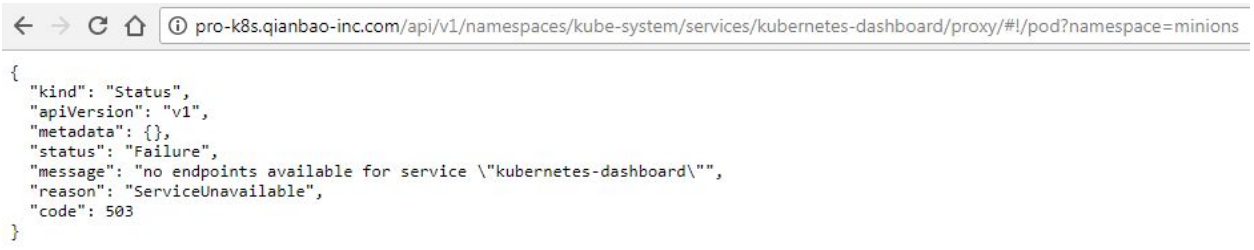
```
<td>URL:</td>
<td>http://pro-k8s.qianbao-inc.com/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy/api/v1/pod/minions?itemsPerPage=10&amp;page=1</td>
</tr>
<tr>
<td>Server:</td>
<td>bjsh19-35-13.qbos.com</td>
</tr>
```

由于Dashboard是通过我们的Nginx访问的，通过加入超时配置，验证是否是Nginx配置导致的，延长了超时时间，问题仍然存在，确认时Dashboard内部 导致的。

```
proxy_read_timeout 600s;
```

仍然超时。

Minions名字空间294个Pods，Dashboard返回503。



删除测试的这些Pods，Dashboard能正常恢复。

3.2.4 问题2：创建Pods时ErrImagePull （无具体影响）

目前测试前已将全部用到的镜像提前拉取到各个节点，不应该出现此问题，但未找到原因，此错误是Docker产生的，此错误存在的时间短，并且重新查看即消失，Pod正常创建。未发现具体影响。

Failed to pull image "reg.qianbao-inc.com/k8s/mmm-minion:latest": rpc error: code = 2 desc = Error response from daemon: {"message":"error unmarshalling content: unexpected end of JSON input"}
Error syncing pod, skipping: failed to "StartContainer" for "minion-3" with ErrImagePull: "rpc error: code = 2 desc = Error response from daemon: {\"message\":\"error unmarshalling content: unexpected end of JSON input\"}"

```
2018-04-25 10:58:16 +0800 CST   2018-04-25 10:58:16 +0800 CST   1
minion-1-1672548023-l2q5d   Pod       spec.containers{minion-1}   Warning
```

```
Failed    kubelet, 10.66.0.11   Failed to pull image
"reg.qianbao-inc.com/k8s/mmm-minion:latest": rpc error: code = 2 desc =
Error response from daemon: {"message":"error unmarshalling content:
unexpected end of JSON input"}
2018-04-25 10:58:16 +0800 CST   2018-04-25 10:58:16 +0800 CST   1
minion-1-1672548023-l2q5d   Pod                 Warning   FailedSync
kubelet, 10.66.0.11   Error syncing pod, skipping: failed to
"StartContainer" for "minion-1" with ErrImagePull: "rpc error: code = 2
desc = Error response from daemon: {\"message\":\"error unmarshalling
content: unexpected end of JSON input\"}"
2018-04-25 10:58:18 +0800 CST   2018-04-25 10:58:18 +0800 CST   1
minion-1-1672548023-l2q5d   Pod                 Warning   FailedSync
kubelet, 10.66.0.11   Error syncing pod, skipping: failed to
"StartContainer" for "minion-1" with ImagePullBackOff: "Back-off pulling
image \"reg.qianbao-inc.com/k8s/mmm-minion:latest\""
```

```
Apr 25 10:51:52 bjsh66-0-11.qbos.com dockerd[2149]:
time="2018-04-25T10:51:52.833437420+08:00" level=info msg="Attempting next
endpoint for pull after error: error unmarshalling content: unexpected end
of JSON input"
Apr 25 10:51:52 bjsh66-0-11.qbos.com dockerd[2149]:
time="2018-04-25T10:51:52.833517759+08:00" level=error msg="Handler for
POST /v1.30/images/create returned error: error unmarshalling content:
unexpected end of JSON input"
```

后期已是频繁现象，发现在一个Deployment的副本（replica数）小于等于2时，无此问题，副本为3和4会出现此现象。

3.3 Docker和虚拟机服务性能对比

3.3.1 测试方法

1. 创建模拟的目标服务
2. 分别部署在Kubernetes和虚拟机下
3. 对不同的服务通过hey进行压测以得到性能数据

服务为一个HTTP的Web服务：
http://issue.qianbao-inc.com/SRE/example-apps/src/branch/master/fs-deployment

具体内容见：

http://issue.qianbao-inc.com/SRE/k8s/src/branch/master/testing/httpload

Httpload为相应的自动化测试

虚拟机配置：4核，8G内存
Pod的配置：4核，8G内存
客户端配置：4核，8G内存 (Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz)

客户端采用同一个客户端（机器配置和测试方法皆为一样），Pod（等同于容器的概念） 跑在该虚拟机之上，即同一台虚拟机，VM直接运行服务，Pod通过Docker运行服务。

3.3.2 测试结果

运行VM的服务的性能，与运行在Kubernetes Docker上服务的性能基本接近，无显著的区别。

测试过程中有遇到如下问题：

1. Pod性能波动不稳定问题
   测试的总请求数由20000改为了80000，并新增了一百万和一千万的请求测试，新的测试无结果波动大问题。

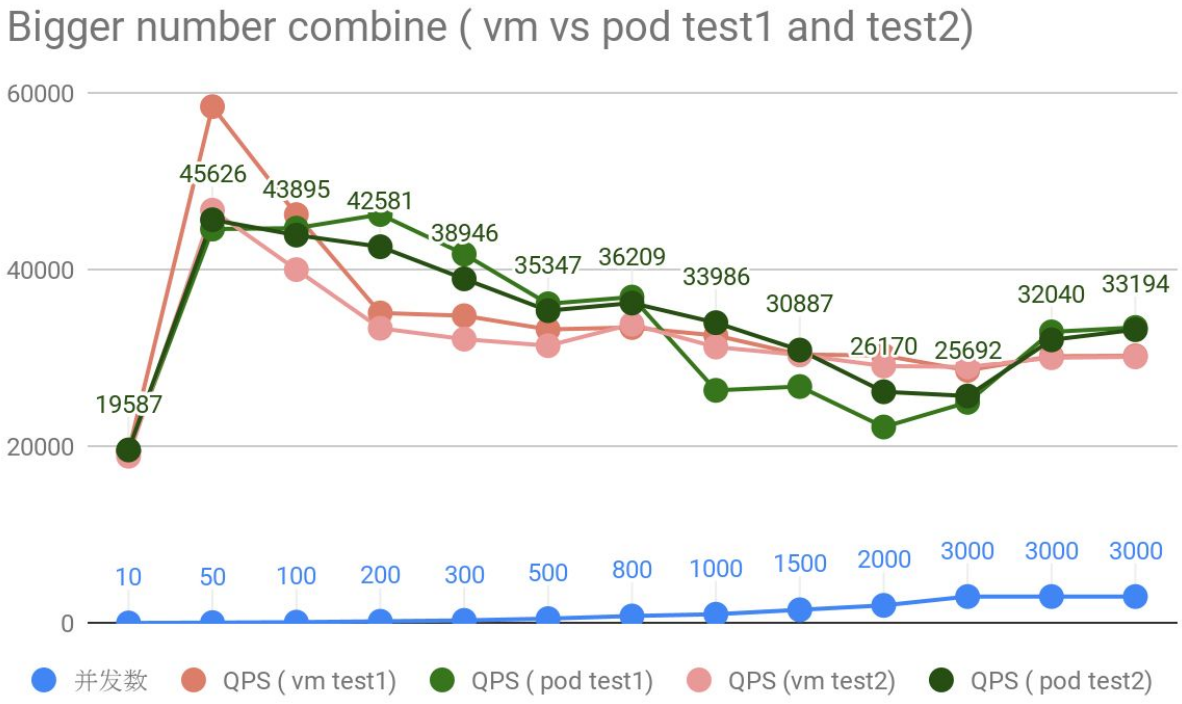
2. Pod的和VM性能测试结果相差太大
   已排查是由于服务运行的方式，导致HTTP Response Body的大小不一样，即而导致Pod和VM性能测试结果相差太大。

发现Nginx代理3个服务实例，并没有带来明显的性能提升，属于正常，多实例的特点在于能同时处理更多的并发请求。

3.3.3 详细数据

对比结果为VM和Pod性能基本接近，无显著区别。

对比结果图如下（已更新）：

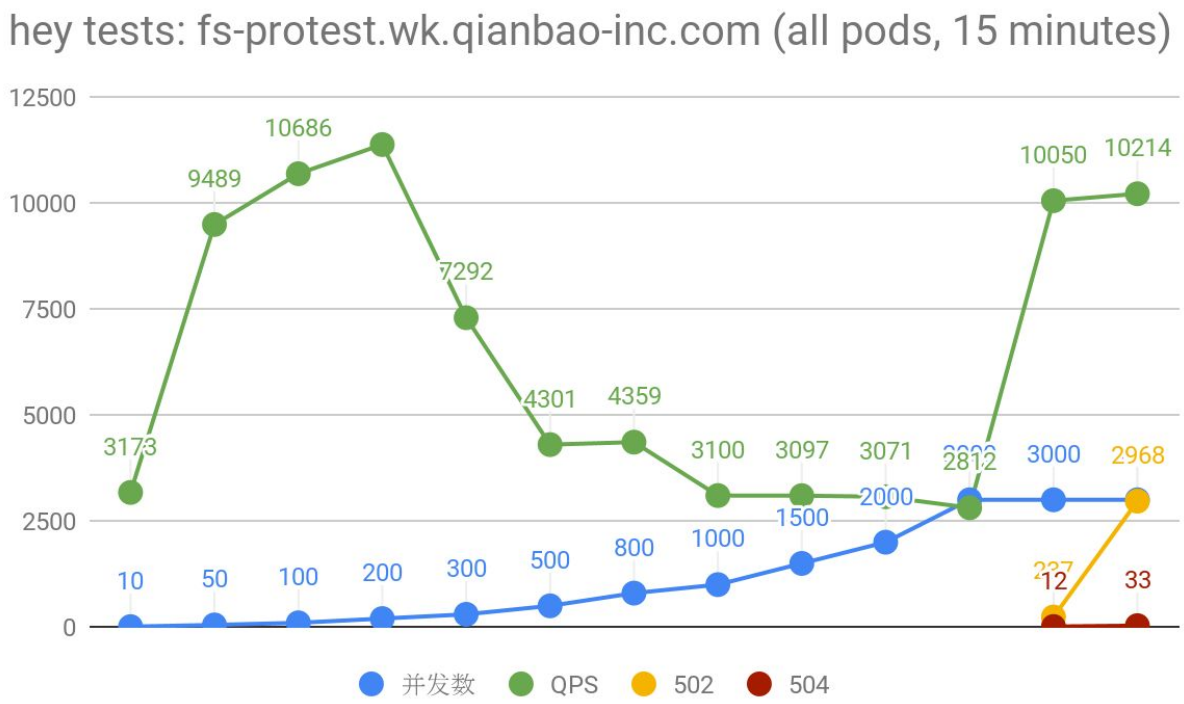


具体数据如下：

服务名	并发数	QPS (vm test1)	QPS (pod test1)	QPS (vm test2)	QPS (pod test2)
fs(N=80000)	10	19419	19457	18875	19587
fs(N=80000)	50	58431	44576	46711	45626
fs(N=80000)	100	46214	44685	39991	43895
fs(N=80000)	200	35087	46227	33351	42581
fs(N=80000)	300	34772	41742	32123	38946
fs(N=80000)	500	33225	36127	31384	35347
fs(N=80000)	800	33440	36884	33816	36209
fs(N=80000)	1000	32539	26329	31230	33986
fs(N=80000)	1500	30388	26762	30338	30887
fs(N=80000)	2000	30269	22184	29082	26170
fs(N=80000)	3000	28590	24973	28983	25692
fs(N=1000000)	3000	30195	32945	30021	32040
fs(N=10000000)	3000	30241	33442	30120	33194

注：其中N为该次测试总的请求数。

通过Nginx访问，测试所有多实例（三个实例），即所有Pods负载均衡(round-robin)访问（hosts绑定Nginx服务器一台）。



（更新2018-5-10）上图Nginx存在问题，为排除Nginx本身的问题影响对比结果，以下用Gobetween作为代理再次进行对比的数据：

3.3.4 问题1：性能波动较大（已解决）

通过直接Pod访问和从Nginx访问都得出性能波动较大的结果，相比虚拟机的服务性能指标比较稳定 。

更新：已解决，新的测试结果无此问题，测试图已更新。

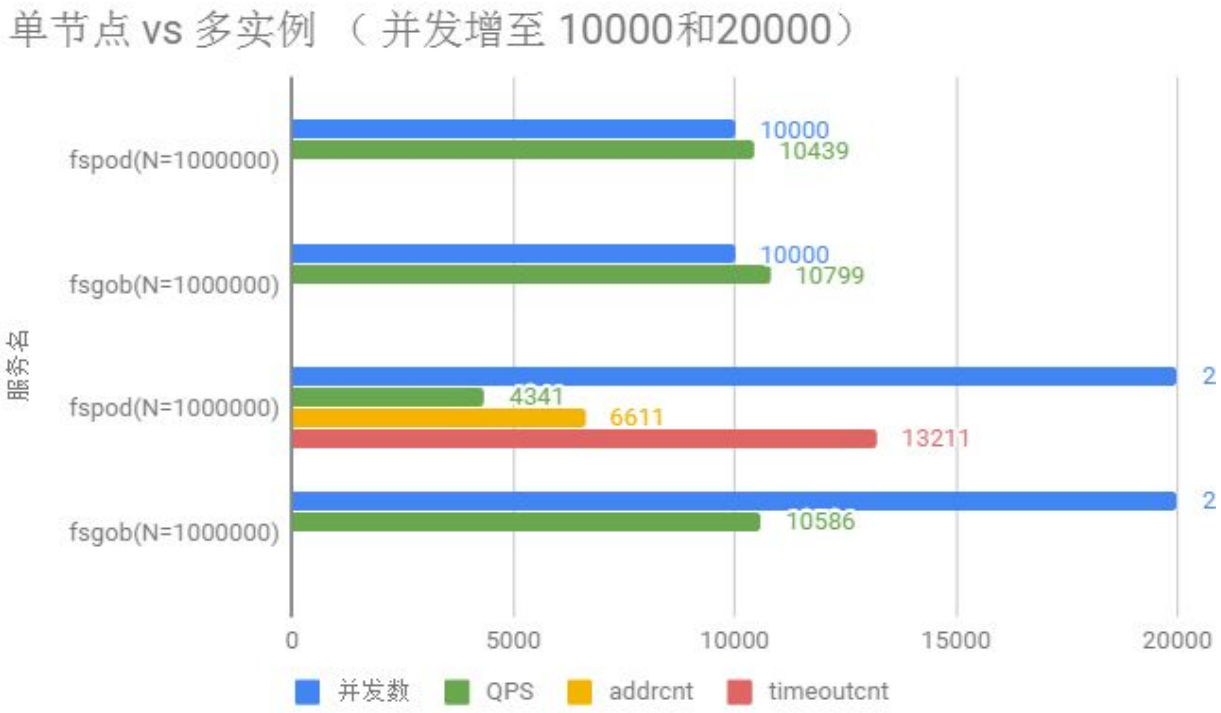
3.3.5 问题2：多实例未带来性能提升（已解决）

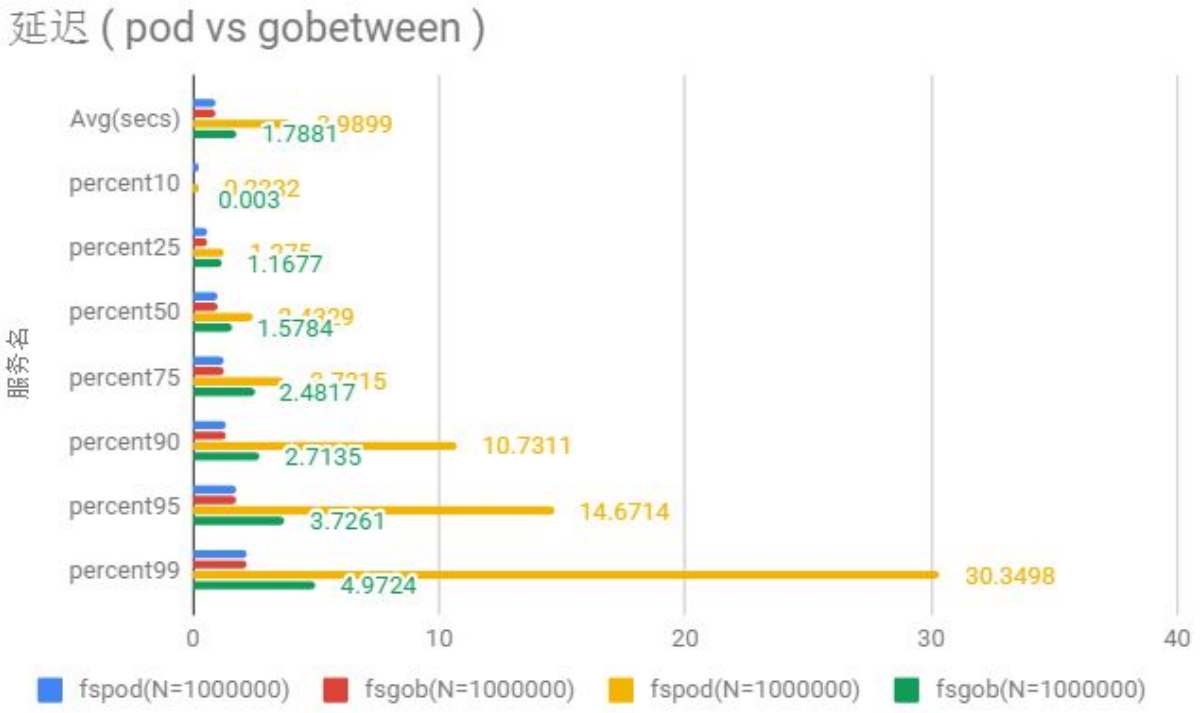
通过上图发现多实例未带来直接的性能提升？此测试有待进一步观察，及验证测试的方法是否有误。

更新（2018-5-10 ）：

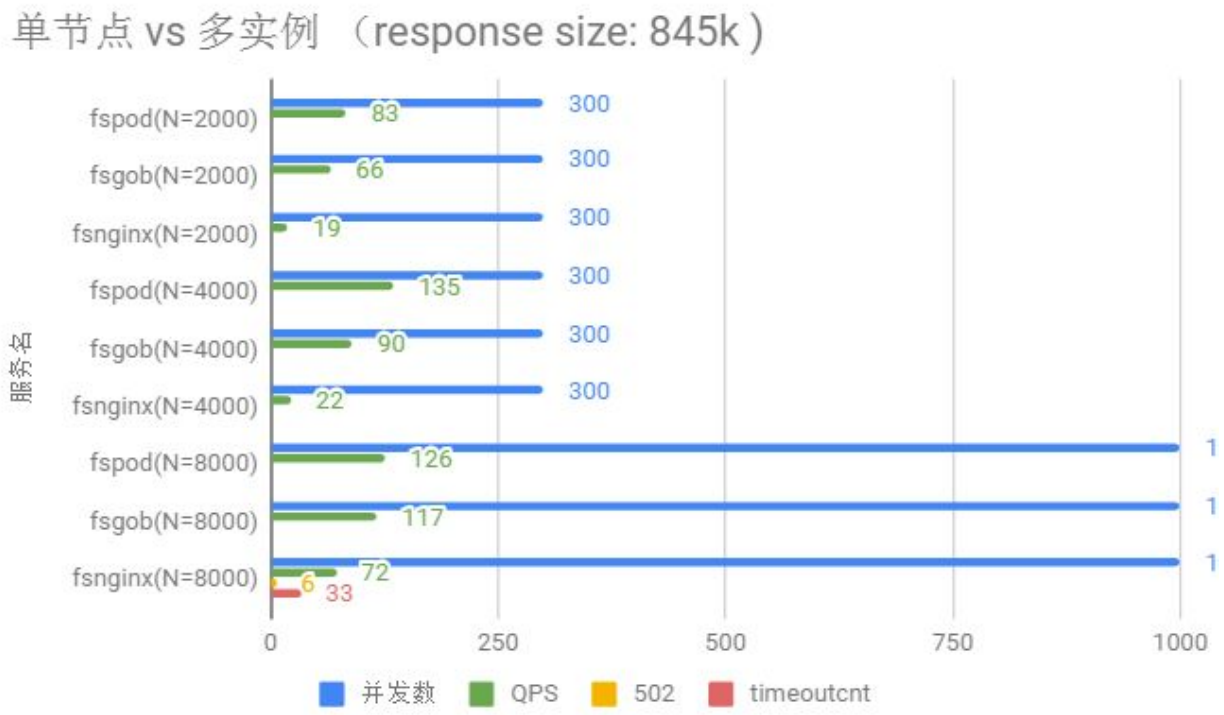
根据测试的结果，及了解到的信息，多节点没有性能提升是正常的，由此试着发现多实例能带来什么效果？

目前发现多实例能处理更多的并发请求，单实例在并发量到20000时，多实例处理能力与之前持平，单实例处理能力低于平时范围。





同时发现在HTTP请求返回的数据量大时，经过Nginx相关代理的性能低于节点直接访问。



4 环境信息

4.1 生产环境

软件信息：

Node1 (10.66.0.10):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 4.4.105-1.el7.elrepo.x86_64
- Docker: client & server (17.06.2-ce, API version 1.30)
- Cpu cores: 32
- Memory: 257838M (~256G)
- Disk free space: 3.8T (/apps/docker)

Node2 (10.66.0.11):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 4.4.105-1.el7.elrepo.x86_64
- Docker: client & server (17.06.2-ce, API version 1.30)
- Cpu cores: 32
- Memory: 257838M (251G)
- Disk free space: 3.8T (/apps/docker)

Node3 (10.66.0.12):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 4.4.105-1.el7.elrepo.x86_64
- Docker: client & server (17.06.2-ce, API version 1.30)
- Cpu cores: 32
- Memory: 257838M (251G)
- Disk free space: 3.8T (/apps/docker)

Node4 (10.66.0.18):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 4.4.105-1.el7.elrepo.x86_64
- Docker: client & server (17.06.2-ce, API version 1.30)
- Cpu cores: 32
- Memory: 128814M (127G)
- Disk free space: 2.7T (/apps/docker)

Master1 (10.66.8.168):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 3.10.0-514.21.2.el7.x86_64
- Docker: not installed
- Cpu cores: 4 (Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz)
- Memory: 7565M (7G)
- Disk free space: 57G (/)

Master2 (10.66.8.169):

- OS: CentOS Linux release 7.3.1611 (Core)
- Kernel: 3.10.0-514.21.2.el7.x86_64
- Docker: not installed
- Cpu cores: 4 (Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz)
- Memory: 7565M (7G)
- Disk free space: 57G (/)

Virtual IP 10.66.8.201

Kubernetes cluster version: v1.6.13

- Master1: kube-controller-manager, kube-scheduler, kube-apiserver
    - keepalived (v1.3.5-1), haproxy (v1.5.18-6)
- Master2: kube-controller-manager, kube-scheduler, kube-apiserver
    - keepalived (v1.3.5-1), haproxy (v1.5.18-6)

Kubernetes相关组件版本为v1.6.13发行版本（新编译的kube-apiserver修复apiserver到etcd连接超时的bug）。

l旧版本为Kubernetes v1.6.8。

4.2 测试环境

软件信息：

Node1 (172.28.46.123):

- OS: Centos 7.3.1611
- Kernel: 3.10.0-514.el7.x86_64
- Docker: client (17.05.0-ce, API version 1.29), server (17.05.0-ce, API version 1.29)

Node2 (172.28.46.124):

- OS: Centos 7.3.1611
- Kernel: 4.4.105-1.el7.elrepo.x86_64
- Docker: client (17.05.0-ce, API version 1.29), server (17.05.0-ce, API version 1.29)

Node3 (172.28.46.125):

- OS: Centos 7.2.1511
- Kernel: 3.10.0-327.el7.x86_64
- Docker: client (17.05.0-ce, API version 1.29), server (17.05.0-ce, API version 1.29)

Kubernetes cluster version: v1.6.8

- Node1: kube-controller-manager, kube-scheduler, kube-apiserver
    - kube-proxy, kubelet, docker
- Node2: kube-proxy, kubelet, docker
- Node3: kube-proxy, kubelet, docker

Kubernetes相关组件版本为v1.6.8发行版本（新编译的kube-apiserver修复apiserver到etcd连接超时的bug）。