ceph

Ceph中国社区 | IT大咖说 知识共享平台

CEPHALOCON APAC 2018

THE FUTURE OF STORAGE

22-23 March 2018 | BEIJING

Doing Quality of Service without QoS

David Byte
Sr. Technical Strategist
SUSE

Alex Lau
Storage Consultant
SUSE

The Challenge

Many customers desire Quality of Service.

- Traditional storage provides it
- Modern storage needs it

Current State of Aff...

There isn't a mechanism in place for providing QoS today and the ISCSI target providers don't support this directly either.

There are multiple way to provide different forms of QoS.

- The client can limit their own read/write/iops.
- Control traffic at the gateway
- Traffic shaping via the network
- Ceph Native QoS

Upstream effort

In the upstream community, work is ongoing to implement QoS. But a distributed storage QoS is not easy to do. It also involved with the dmclock implementation.

https://github.com/ceph/dmclock
https://github.com/ceph/ceph/pull/17450

Possible solutions fc. ..ew

For RGW, load-balancers may provide some functionality

For other protocols, there isn't much…

iscsi manipulate cmdsn_depth queue

tc (Traffic Control) is built into the Linux kernel and is able to provide weighted queues similar to network QoS.

- Option 1 – bandwidth cap
- Option 2 – inject latency

Adjust ISCSI cmdsn_depth queue

This is not really QoS, but queue depth controls max I/O

- 1 queue depth size = slowest
- 64 = much faster

Pro:
Simple script to automate

Con:
Not exact
Minimum may still be too high
Adjustment by hand is still necessary

cmdsn_depth Sampl

First we need to get the target and initiator name
e.g. /sys/kernel/config/target/iscsi/{target}
/tpgt_1/acls/{initiator}

```
if [ ! $1 ]; then
        echo "Please provide target name to adjust speed"
        exit -1
fi
TARGET=$1
if [ ! $2 ]; then
        echo "Please provide initiator name to adjust
speed"
        exit -1
fi
INITIATOR=$2
```

Check target and ACL

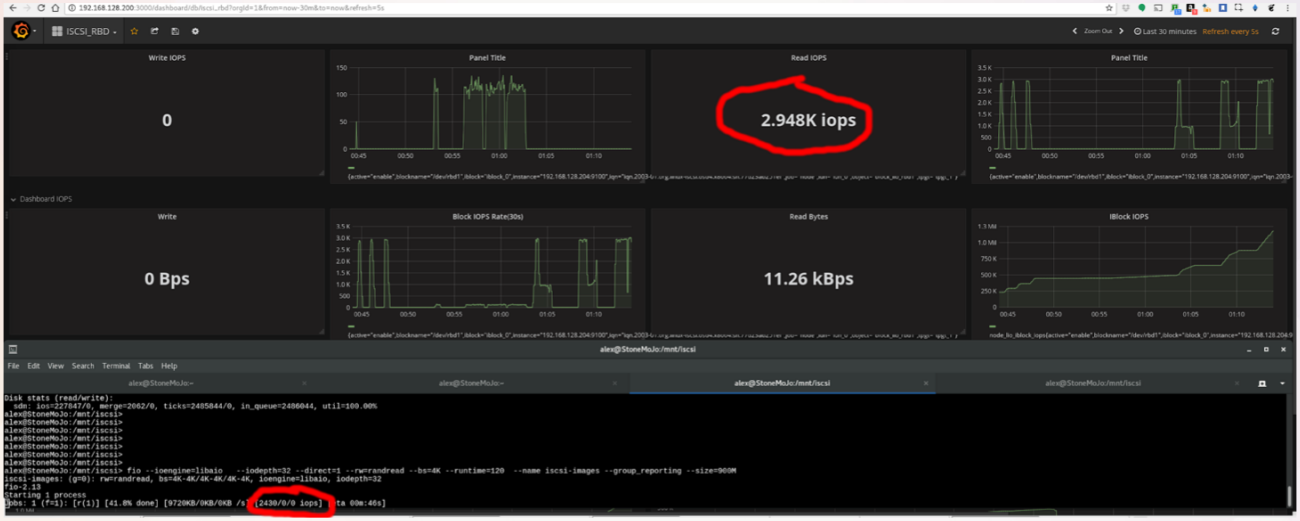

```
# Check target exist and Check ACL is enable
TARGET_PATH=/sys/kernel/config/target/iscsi/$TARGET
CMD_DEPTH_PATH=$TARGET_PATH/tpgt_1/acls/$INITIATOR/cmdsn_depth

if  [ ! -f $TARGET_PATH ]; then
        if [ ! -d $TARGET_PATH/tpgt_1/acls ]; then
                echo "Target need acl to allow throlle to work"
                exit -1
        else
                if [ ! -f $CMD_DEPTH_PATH ]; then
                        echo "Initiator throttler controller
doesn't exist"
                        exit -1
```

Script to set cmdsn_


```
echo "Please enter [min max] to adjust speed ?"
        select result in "min" "max"; do
                case $result in
                        "min" ) echo 1 > $CMD_DEPTH_PATH ;
                        echo "Now $INITIATOR running at slowest speed"
                        break;;
                                        "max" ) echo 64 > $CMD_DEPTH_PATH ;
                        echo "Now $2 should be running at fastest speed"
                                        break;;
                                esac
                done
```

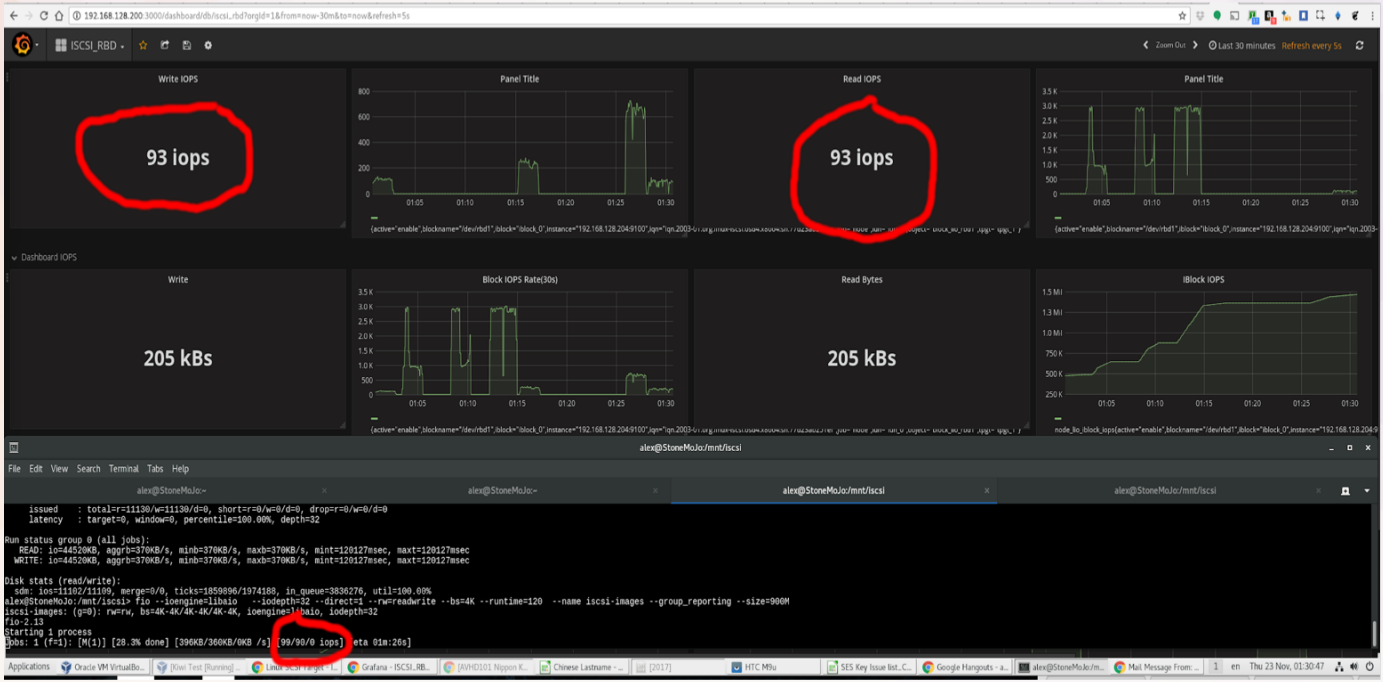
How did I monitor the

With openATTIC and Grafana Prometheus and node_exporter, we can monitor the iscsi read, write and ops more easily.



However the module still currently in PR waiting to get into prometheus.
https://github.com/prometheus/procfs/pull/69
https://github.com/prometheus/node_exporter/pull/776

After dropping it to cm



Use tc to control bandwidth

Can filter based on source IP address or target IP address

tc qdisc add dev eth0 root handle 1: htb default 30
tc class add dev eth0 parent 1: classid 1:1 htb rate 10000mbit burst 15m
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5000mbit burst 15m
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3000mbit burst 15m
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 100mbit ceil 9000mbit burst 15m
The author then recommends SFQ for beneath these classes:
tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
#Filter based on destination (iscsi target) IP
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip dst 4.3.2.1/32 flowid 1:10
#Filter based on source (iscsi initiator) IP
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip src 1.2.3.4/32 flowid 1:10

Use tc to inject latency


```
tc qdisc add dev eth0 root handle 1: prio
tc qdisk add dev  eth0 parent 1:1 handle 10: netem delay .05ms

#Filter based on destination (iscsi target) IP
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip dst 4.3.2.1/32 flowid 1:1

#Filter based on source (iscsi initiator) IP
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip src 1.2.3.4/32 flowid 1:1
```

tc Methods Pros &

Pros:
Better control for bandwidth
Easily managed through salt or ansible

Cons:
tc is complex
Not the easiest to use (hundreds of clients = high complexity)
It doesn't control IOPS
Packets can get dropped

Thoughts:
Use multiple subnets for ISCSI initiators. Each subnet has it's own filter and thus QoS setting. This only makes sense with injected delays

Our thoughts and recommendations

If possible, wait for upstream to provide a ceph native solution.

If not, carefully select, test, and implement a solution that works for your particular use case.