Deterministic Storage Performance

'The AWS way' for Capacity Based QoS with OpenStack and Ceph

Kyle Bader - Senior Solution Architect, Red Hat
Sean Cohen - A. Manager, Product Management, OpenStack, Red Hat
Federico Lucifredi - Product Management Director, Ceph, Red Hat

May 2, 2017
Block Storage QoS in the public cloud
WHY DOES IT MATTER?
It’s what the user wants

Every Telco workload in OpenStack today has a DBMS dimension to it

QoS is an essential building block for DBMS deployment

Public Cloud has established capacity-based QoS as a de-facto standard
PROBLEM STATEMENT
Deterministic storage performance

- Some workloads need deterministic performance from block storage volumes
- Workloads benefit from Isolation from “noisy neighbors”
- Operators need to know how to plan capacity
Blocks storage in a public cloud

The basics

- **Ephemeral / Scratch Disks**
  - Local disks connected directly to hypervisor host

- **Persistent Disks**
  - Remote disks connected over a dedicated network

- Boot volume type depends on instance type
- Additional volumes can be attached to an instance
THE AWS WAY
Elastic Block Storage

- AWS EBS
  - EBS-backed instances
  - SSD-backed volumes
  - HDD-backed volumes
- Dynamically re-configurable at runtime
  - Mount (boot or runtime)
  - Resize
- Monitoring
  - CloudWatch metrics
- Automation
  - CloudFormation

![Graphs showing read and write bandwidth and throughput over time.](#)
EBS Volumes: an example

General purpose SSD

- I/O Provisioned gp2 volume
  - Baseline: 100 IOPS
    - + 3 IOPS per GB (up to 10,000 IOPS)
  - Burst: 3,000 IOPS (up to 1 TB)
  - Thruput: 160 MB/s
  - Latency: single-digit ms
  - Capacity: 1 GB to 16 TB
THE AWS WAY
Elastic Block Storage

- **Flavors**
  - Magnetic ~100 IOPS and 40 MB/s per volume
  - General Purpose SSD (3 IOPS/GB)
  - Provisioned IOPS (30 IOPS/GB)

- **Elastic Volumes**
  - gp2, io1, st1, scl volume types
  - Increase volume size (cannot shrink!)
  - Change provisioned IOPS
  - Change volume type

- Single dimension of provisioning: amount of storage also provisions IOPS
THE GOOGLE WAY

Persistent Disk

- Google Compute
  - Baseline + capacity-based IOPS model
  - Can resize volumes live
  - IOPS and throughput limits
    - Instance limits
    - Volume limits

- Media types
  - Standard Persistent Disk - Spinning Media (0.75r/1.5w IOPS/GB)
  - SSD Persistent Disk - All Flash (30 IOPS/GB)
WHY

We can build you a private cloud like the big boys’

- AWS EBS provides a deterministic number of IOPS based on the capacity of the provisioned volume with Provisioned IOPS. Similarly, the newly announced throughput optimized volumes provide deterministic throughput based on the capacity of the provisioned volume.

- Flatten two different scaling factors into a single dimension (GB / IOPS)
  - Simplifies capacity planning for the operator
  - Operator increases the available capacity by adding more to distributed backend
    - more nodes, more IOPS, fixed increase in capacity

- Lessens the user’s learning curve for QoS
  - Meet users expectations defined by ‘The’ Cloud
Block Storage QoS in OpenStack
OPENSTACK FRAMEWORK TRENDS

What are users running on their clouds?

- % using other
- % using RAILS
- % using WISA
- % using MEAN
- % using Java
- % using LAMP

April 2016 vs October 2015
OPENSTACK CINDER DRIVER TRENDS

Which backend are used in production?
The Road to Block Storage QoS in Cinder

- Generic QoS at hypervisor was first added in Grizzly
- Cinder and Nova QoS support was added in Havana
- Stable API starting Icehouse and ecosystem drivers velocity
- Horizon support was added in Juno

- Introduction of Volume Types, classes of block storage with different performance profiles
- Volume Types configured by OpenStack Administrator, static QoS values per type.
Block Storage QoS in Cinder - Ocata release

**Frontend:** Policy applied to Compute, Limit by throughput
- Total bytes/sec, read bytes/sec, write bytes/sec
- Frontend: Limit by IOPS
- Total IOPS/sec, read IOPS/sec, write IOPS/sec

**Backend:** Policy applied to Vendor specific fields
- HP 3PAR (IOPS: min, max; BWS: min, max, latency, priority)
- Solidfire (IOPS: min, max, burst)
- NetApp (QoS Policy Group) through extra specs
- Huawei (priority) defined through extra specs

### Cinder QoS (throughput based)

<table>
<thead>
<tr>
<th></th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{vendor:disk_type=SSD,</td>
<td>{}</td>
<td>{volume_backend_name=lvm}</td>
</tr>
<tr>
<td></td>
<td>vendor_thick_provisioned=True}</td>
<td>{total_iops_sec=500}</td>
<td>{total_iops_sec=100}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Deployers may optionally define the variable `cinder_qos_specs` to create qos specs.

- `cinder volume-types` may be assigned to a qos spec by defining the key `cinder_volume_types` in the desired qos spec dictionary.
QoS values in Cinder currently are able to be set to static values.

Typically exposed in OpenStack Block Storage API in the following manner:

- **minIOPS** - The minimum number of IOPS guaranteed for this volume. (Default = 100)
- **maxIOPS** - The maximum number of IOPS allowed for this volume. (Default = 15,000)
- **burstIOPS** - The maximum number of IOPS allowed over a short period of time. (Default = 15,000)
- **scaleMin** - The amount to scale the minIOPS by for every 1GB of additional volume size.
- **scaleMax** - The amount to scale the maxIOPS by for every 1GB of additional volume size.
- **scaleBurst** - The amount to scale the burstIOPS by for every 1GB of additional volume size.
- Examples:
  - SolidFire driver in Ocata can recognize 4 QoS spec keys to allow specify settings which are *scaled by the size* of the volume:
    - ‘ScaledIOPS’ a flag used to tell the driver to look for ‘scaleMin’, ‘scaleMax’ and ‘scaleBurst’ which provide the scaling factor from the minimum values specified by the previous QoS keys (‘minIOPS’, ‘maxIOPS’, ‘burstIOPS’).
  - ScaleIO driver in Ocata QoS keys examples:
    - maxIOPSPerGB and maxBWSperGB used.
      - maxBWSperGB - the QoS I/O bandwidth rate limit in KBs.
      - The limit will be calculated by the specified value multiplied by the volume size.
QoS values in Cinder currently are able to be set to static values. What if there was a way to derive QoS limit values based on volume capacities rather than static values....
Capacity Derived IOPs

New in Pike release

- A new mechanism to provision IOPS on a per-volume basis with the IOPS values adjusted based on the volume's size (IOPS per GB)
- Allowing OpenStack Operators to cap "usage" of their system and to define limits based on space usage as well as throughput, in order to bill customers and not exceed limits of the backend.
- Associating IOPS and size allows you to provide tiers such as:

<table>
<thead>
<tr>
<th>Capacity Based QoS (Generic)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gold</td>
<td>1000 GB at 10000 IOPS per GB</td>
</tr>
<tr>
<td>Silver</td>
<td>1000 GB at 5000 IOPS per GB</td>
</tr>
<tr>
<td>Bronze</td>
<td>500 GB at 5000 IOPS per GB</td>
</tr>
</tbody>
</table>
Capacity Derived IOPs

Cinder QoS API - New Keys

- Allow creation of qos_keys:
  - read_iops_sec_per_gb
  - write_iops_sec_per_gb
  - total_iops_sec_per_gb

- These functions are the same as our current <x>_iops_sec keys, except they are scaled by the volume size.

<table>
<thead>
<tr>
<th>QoS Spec Key</th>
<th>QoS Spec Value</th>
<th>2 GB Volume</th>
<th>5 GB Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read IOPS / GB</td>
<td>10000</td>
<td>20000 IOPS</td>
<td>50000 IOPS</td>
</tr>
<tr>
<td>Write IOPS / GB</td>
<td>5000</td>
<td>10000 IOPS</td>
<td>25000 IOPS</td>
</tr>
</tbody>
</table>
THEORY OF STORAGE QOS
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL
SCALE - I do not think it means what you think it means
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means

Contention + Coherency Delay
UNIVERSAL SCALABILITY MODEL

SCALE - I do not think it means what you think it means

Contention + Coherency Delay

This is normal, everything is fine.
DISK BASED CLUSTERS
Higher coherency delay due to seeking

- Diminishing returns from contention
- Negative returns from incoherency
SSD BASED CLUSTERS
Lower coherency delay, no seeks

- Diminishing returns from contention
- Negative returns from incoherency (marginal)
SCALING DIMENSIONS

What scales?

- Increase height of each block with faster media - IOPS limit
- Increase number of blocks by adding more OSD hosts - Volume quota
- Volume quota less relevant for SSDs, low coherency delay
CAPACITY BASED LIMITS

Brought to you by low latency media!

- More small volumes with low iops
- Less large volumes with high iops
- Mix and match
RESULTS
Disaggregated Volume Scalability (librbdfio 16KB randrw)
RESULTS

Hyperconverged Volume Scalability (librbd/fio 16KB randrw)
RESULTS

Hyperconverged Volume Scalability (kvmrbdpio 16KB randrw)
SUMMARY OF RESULTS

What does it mean to me?

- OSD - 7.2k RPM, writeback cache, SSD journal
- One volume per OSD
- 100 IOPS per volume
- Flash - 500 IOPS per OSD GHz - Intel P3700
The future
CEPH + OPENSTACK QOS

Where are we, where are we going?

- Magnetic style block volumes - fixed IOPS per volume*
- Provisioned IOPS style block volumes - scaled IOPS per GB*
- General purpose SSD - work in progress, distributed QoS implementation required

* with capacity planning
OPENSTACK TOOLS AND GAPS

- **Monitoring**
  - Telemetry - via Gnocchi plugin for Grafana dashboard
  - QEMU
    - There's an interface in QEMU to request block stats ("info blockstats" command), also exposed via libvirt but not yet in OpenStack
  - Ceph - RBD client stats socket
  - Event triggering automation (see AWS CloudWatch example)

- **Elasticity**
  - Change volume types limits
  - You can make the volumes larger (hot-grow) but not shrink them
  - Dynamically re-configurable at runtime
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.