СЛЁРМ
+
Southbridge

Архитектура Ceph

Виталий Филиппов

СЛЁРМ Ceph Видеокурс

Архитектура Ceph

Виталий Филиппов

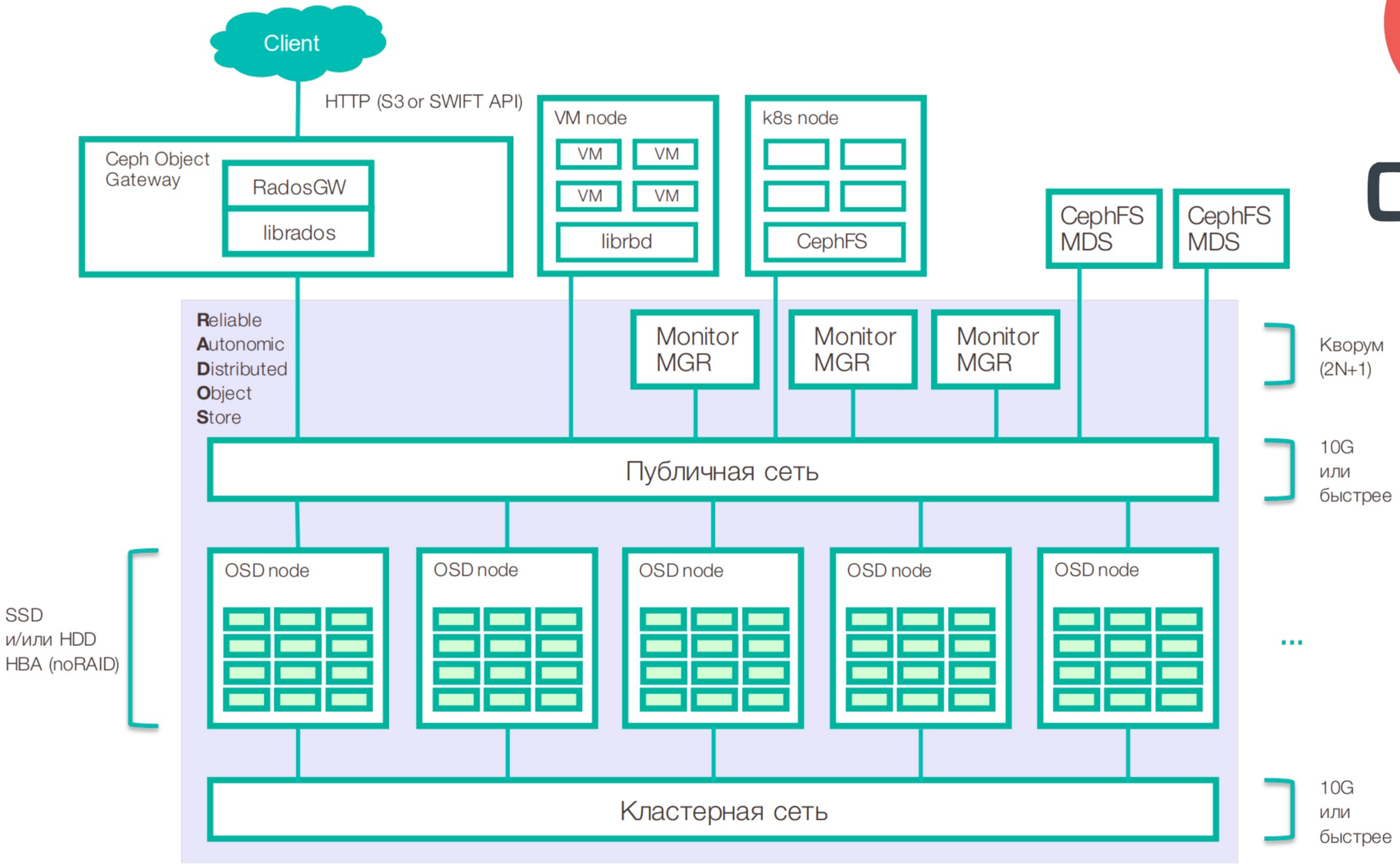
Разработчик-эксперт в компании CUSTIS, линуксоид. Занимаюсь разработкой на разных языках от node.js до C++, сильно упоролся по Ceph-у. :)

Автор статьи «Производительность Ceph»

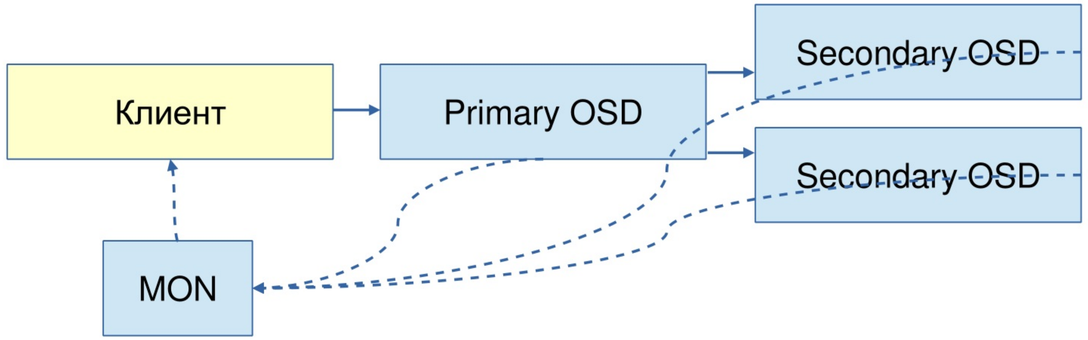
Общие замечания

1. Достаточно сложная система, ~1 миллион строк кода. Интересная. Но есть чему ломаться. :-)
2. OSD, Monitor, Manager
3. RADOS и CRUSH. Протоколы поверх RADOS
4. Bluestore и Filestore
5. RGW (RADOS GateWay), MDS (MetaData Server)
6. Системы установки и оркестрации, клиентские библиотеки

Обзор архитектуры Ceph



OSD



1. OSD — Object Storage Daemon.
   Хранит все данные в виде «объектов».

2. Во многих других системах клиенты обращаются к данным через единую точку входа и она быстро становится узким местом.

3. В Ceph клиенты работают **напрямую с OSD**.
   Архитектура симметричная, все OSD равноправны.

4. Распределение данных основано на консистентном хешировании, поэтому клиенты сразу знают, на какие OSD идти за данными.

5. Primary-репликация. При записи клиент отправляет только 1 запрос на OSD. Остальные копии создаёт уже не клиент, а первичный OSD.

Мониторы (ceph-mon)

1. Хранят карту кластера: **monmap** (перечень мониторов), **osdmap** (перечень OSD), **crushmap** (правила репликации), а также ключи аутентификации, в том числе ключи самих OSD

2. Мониторы образуют консенсус (алгоритм PAXOS), система переживает потерю до (N-1)/2 мониторов. При 3 или 4 мониторах без даунтайма может упасть 1, при 5 или 6 — 2 и т.п.

3. Мониторов должно быть > 1, на разных дисках! Терять их не надо! (хотя потеря всех мониторов — не катастрофа, копии карт есть в OSD, их можно восстановить).

4. Мониторы хранят данные в ФС /var/lib/ceph/mon/*/ в RocksDB.

Мониторы — требования

1. Мониторам нужен NTP (синхронизация часов) для нормальной работы, ибо PAXOS.

2. На флешках размещать мониторы нельзя :) карты постоянно меняются, если тормозят мониторы — тормозит весь Ceph.

3. Нагрузка на мониторы прямо пропорциональна числу OSD.

4. Потребление памяти обычно ~1-2 ГБ на демон.

5. На системном диске (там же, где ОС) размещать мониторы можно.

6. Мониторы можно совмещать с OSD, пока кластер небольшой и есть запас по CPU и сети.

Менеджеры (ceph-mgr)

1. Дополнение к мониторам, обязательны начиная с Ceph 12 Luminous
2. Написаны на Python и C++, реализуют некритичные функции управления: dashboard, мониторинг, автовыбор числа PG, балансировку PG и т. п.
3. Данных не хранят
4. Ставятся обычно рядом с мониторами (но можно ставить куда угодно)
5. Если не работают, ceph -s показывает ерунду
6. Потребляют немного памяти (< 1 ГБ на демон)

RADOS

1. RADOS — базовый объектный протокол. Все данные хранятся в виде объектов. ID объекта — строка.

2. Reliable Autonomic Distributed Object Store — придумали аббревиатуру, а потом расшифровали :)

3. RADOS не очень-то простой: записи атомарны, есть поддержка случайной перезаписи и хранения key-value метаданных (xattr и omap)

4. С RADOS можно работать и напрямую, цена вопроса — 50 строк кода. Однако есть ограничения: например, размер объекта — до 128 МБ.

5. Поэтому в реальности используются интерфейсы, работающие поверх RADOS.

Интерфейсы поверх RADOS

1. RBD — RADOS Block Device, хранилище для дисков виртуальных машин. Образы режутся на 4 МБ(*) объекты и сохраняются в RADOS.

2. Интерфейс RBD реализован на клиенте, специального “сервера RBD” нет. С RBD умеет напрямую работать qemu/kvm.

3. CephFS — файловая система, требует дополнительный сервер метаданных — MDS (MetaData Server).

4. RGW — RADOS GateWay, предоставляет S3-совместимое хранилище.

5. Есть реализации iSCSI (tgt/lio), NBD поверх RBD.
   Есть реализации Samba VFS, NFS (Ganesha) поверх CephFS.

CRUSH

1. Выбор OSD задаётся алгоритмом CRUSH — Controlled Replication Under Scalable Hashing *(тоже сначала придумали аббревиатуру, а потом расшифровали)*

2. Объекты группируются в PG (placement groups). PG — единица распределения данных. Все объекты одной PG хранятся на одном и том же наборе OSD (по умолчанию — 3 реплики, то есть 3 OSD).
   PG объединяются в Pool-ы.

3. OSD для PG выбираются псевдослучайно с помощью straw* хеша (разновидность rendezvous хеша) так, чтобы число PG на OSD получалось приблизительно пропорционально его весу (объёму).
   $\max_{osd=1..n}(weight_{osd} * hash(pg\_number))$

   * *straw, потому что OSD как будто тянут соломинки*

Котята

Котёнок приходит в кошачье общежитие из нескольких (не менее 3) корпусов.

Тут же материализуется ещё две копии котёнка, которые разбредаются по псевдослучайно выбранным комнатам.

Из-за косяков алгоритма некоторые комнаты наполняются быстрее других и тогда котят приходится разгонять по другим комнатам для более однородного заполнения.

Если переполнится хотя бы одна комната, общежитие вообще перестаёт пускать новых котят, пока не построят новый корпус общежития.

CRUSH-дерево

OSD объединены в древовидную структуру “crush tree”. Всё, что выше OSD (host, rack, datacenter и т.п.) называется “crush bucket”.

```
# ceph osd df tree

ID   CLASS  WEIGHT    REWEIGHT  SIZE     RAW USE  DATA     OMAP     META      AVAIL    %USE  VAR   PGS  STATUS  TYPE NAME
 -1         38.20674         -   38 TiB  333 GiB  321 GiB   26 MiB    12 GiB   38 TiB  0.85  1.00    -          root default
-31         38.20674         -   38 TiB  333 GiB  321 GiB   26 MiB    12 GiB   38 TiB  0.85  1.00    -              rack sill-hdd
 -3         12.73558         -   13 TiB  110 GiB  106 GiB  9.2 MiB   4.0 GiB   13 TiB  0.85  0.99    -                  host sill-01
  0    ssd   1.81940   1.00000  1.8 TiB   15 GiB   14 GiB  858 KiB  1023 MiB  1.8 TiB  0.82  0.96   89      up              osd.0
  1    ssd   1.81940   1.00000  1.8 TiB   17 GiB   16 GiB  1.4 MiB  1023 MiB  1.8 TiB  0.92  1.08   92      up              osd.1
  2    ssd   7.27739   1.00000  7.3 TiB   62 GiB   61 GiB  6.1 MiB  1018 MiB  7.2 TiB  0.83  0.97  323      up              osd.2
  4    ssd   1.81940   1.00000  1.8 TiB   16 GiB   15 GiB  803 KiB  1023 MiB  1.8 TiB  0.86  1.01   89      up              osd.4
-16         12.73558         -   13 TiB  110 GiB  106 GiB  7.9 MiB   4.0 GiB   13 TiB  0.85  0.99    -                  host sill-02
  5    ssd   1.81940   1.00000  1.8 TiB   13 GiB   12 GiB  455 KiB  1024 MiB  1.8 TiB  0.69  0.81   85      up              osd.5
  7    ssd   7.27739   1.00000  7.3 TiB   64 GiB   63 GiB  3.6 MiB  1020 MiB  7.2 TiB  0.87  1.02  340      up              osd.7
  9    ssd   1.81940   1.00000  1.8 TiB   17 GiB   16 GiB  2.7 MiB  1021 MiB  1.8 TiB  0.93  1.09   92      up              osd.9
 11    ssd   1.81940   1.00000  1.8 TiB   16 GiB   15 GiB  1.1 MiB  1023 MiB  1.8 TiB  0.84  0.98   76      up              osd.11
-13         12.73558         -   13 TiB  113 GiB  109 GiB  8.8 MiB   4.0 GiB   13 TiB  0.87  1.02    -                  host sill-03
  3    ssd   1.81940   1.00000  1.8 TiB   15 GiB   14 GiB  928 KiB  1023 MiB  1.8 TiB  0.83  0.98   91      up              osd.3
  6    ssd   7.27739   1.00000  7.3 TiB   68 GiB   67 GiB  5.3 MiB  1019 MiB  7.2 TiB  0.91  1.07  345      up              osd.6
  8    ssd   1.81940   1.00000  1.8 TiB   14 GiB   13 GiB  892 KiB  1023 MiB  1.8 TiB  0.75  0.88   80      up              osd.8
 10    ssd   1.81940   1.00000  1.8 TiB   15 GiB   14 GiB  1.8 MiB  1022 MiB  1.8 TiB  0.82  0.96   85      up              osd.10
                       TOTAL    38 TiB  333 GiB  321 GiB   26 MiB    12 GiB   38 TiB  0.85
MIN/MAX VAR: 0.81/1.09  STDDEV: 0.07
```

CRUSH-правила (rules)

1. CRUSH-правила задают возможный выбор OSD по дереву

2. Можно (но не нужно) править руками

```
ceph osd getcrushmap -o crushmap; crushtool -d crushmap > crushmap.txt; (правим);
crushtool -c crushmap.txt -o crushmap; ceph osd setcrushmap -i crushmap
```

3. Обычно только выбираете тип, число реплик / устройств данных и чётности, корень, класс устройства и домен отказа. Шпаргалка:

```
ceph osd crush rule create-replicated <name> <root> <failure-domain> [<device-class>]
```

```
ceph osd erasure-code-profile set <name> k=2 m=1 [crush-root=default] [crush-failure-
domain=osd] [crush-device-class=hdd] [plugin=jerasure|shec|lrc|clay|isa]
```

Pool-ы и правила

1. CRUSH-правило назначается пулу, и каждая его PG получает набор OSD, выбранный по этому правилу:

   ```
   ceph osd pool create <pool_name> <pg_count> [<pgp_count> [replicated|erasure [<rule_name>]]]
   ```

2. Кроме правила у Pool-а есть “size” и “min_size”.
   Число реплик в реплицированных пулах задаёт именно “size”.
   В EC-пулах “size” не меняется.
   ”min_size” — минимальное число реплик, разрешённое для работы пула.
   По умолчанию 3/2 у реплицированных, у EC — k+n/k+1.

3. Правило и size можно менять на ходу, Ceph перенесёт данные без даунтайма:

   ```
   ceph osd pool set <pool_name> crush_rule <rule_name>
   ```

Bluestore и Filestore

1. Слой хранения OSD называется object store. Есть 2 реализации («2 стула»).

2. Filestore: объекты пишутся на диск, отформатированный в XFS, в виде файлов. xattr — в xattr, omap — в RocksDB/LevelDB рядом. Для атомарности — собственная реализация журнала. Журнал можно вынести на другой диск (SSD).

3. Bluestore (**bl**ock+n**ew** store): объекты пишутся на блочное устройство (raw device), смещения, omap и всё остальное в RocksDB, одно и то же устройство хитро делится между данными и RocksDB. RocksDB и её WAL тоже можно вынести на другие диски.

4. Bluestore умеет больше: сжатие, контрольные суммы, CoW, перезапись в EC пулах. О системных требованиях и производительности — позже, в теме «Производительность Ceph».

MDS

1. MDS — сервер метаданных CephFS.

2. Участвует в операциях с файлами (open, rename, unlink, flock и т. п.)

3. (Мета)данные хранит в OSD, в отдельном пуле — по объекту на inode + xattr и omap. Если пользователей > 1, под мету обязательны SSD.

4. Кушает память под кэш. По умолчанию — 1 ГБ, но лимит не жёсткий.

5. Кэшу соответствует журнал, который MDS replay-ит и trim-ит.

6. Шардируется (active-active) и резервируется (active-standby).
   Нужен хотя бы 1 standby, в идеале по 1 standby под каждого active.

RGW

1. RADOS Gateway — S3/Swift-сервер Ceph.
2. Данные и метаданные тоже хранит в пулах на OSD.
3. S3 объекты режутся на 4 МБ части.
4. Перечни объектов хранятся в Bucket Index — в omap нескольких RADOS-объектов (шардов индекса), изменения журналируются в Bucket Index Log.
5. Масштабируется в рамках одного кластера/ДЦ, как обычный HTTP сервис.
6. Поддерживает синхронизацию между кластерами/ДЦ — «multisite», в том числе active-active.

Установка/оркестрация

1. Самое старое и простое: ceph-deploy + ceph-volume для добавления OSD
2. Через Ansible: ceph-ansible
3. В k8s: Rook (он вам не дRook)
4. Новомодная оркестрация в контейнерах: cephadm
5. N более редких вариантов типа DeepSea (через SaltStack), croit.io и т. п.
6. В Proxmox есть встроенный установщик — pveceph

Клиентские библиотеки

1. RADOS: librados
2. RBD: librbd, krbd (драйвер ядра), qemu rbd
3. RGW: radosgw-admin, обычные S3: s3cmd, s5cmd
4. CephFS: libcephfs, cephfs-fuse, mount -t ceph
5. Дополнительные извращения: iSCSI (tgt/lio), rbd-nbd, rbd-fuse, CephFS через NFS-Ganesha, RGW через Ganesha и т.п.

На этом обзор архитектуры закончен,
нужно начинать ставить :)