Pnfs over ostor

Vstorage short overview (from man)

No special hardware requirements. Commodity hardware (SATA/SAS drives, 1Gbit+ Ethernet) can be used to create a storage

Strong consistency semantics. This makes Vstorage suitable for VMs and CTs running on top of it (unlike object storage such as Amazon S3 or Swift)

Built-in replication

Automatic disaster recovery on hard drive or node failures

High availability. Data remain accessible even in case of hard drive or node failures

Optional SSD caching

Data checksumming and scrubbing

Grow on demand. More storage nodes can be added to the cluster to increase

Vstorage components (from man)

Metadata server (MDS). MDSs manage metadata, like file names, and keep control over how files are split into chunks and where the chunks are stored. They also track versions of chunks and ensure that the cluster has enough replicas. An MDS can be run in multiple instances to provide high availability. Besides, MDSs keep a global log of important events that happen in the cluster.

Chunk server (CS). A CS is a service responsible for storing real user data chunks and providing access to these data. A Vstorage cluster must have multiple instances of CSs for high availability

Clients. Virtuozzo Containers and virtual machines can be run natively, i.e. directly from the Vstorage cluster. Or clients can simply mount vstorage (as fuse mount, for example)

Object storage definition.

Object storage (also known as **object-based storage**[1]) is a computer data storage architecture that manages data as objects, as opposed to other storage architectures like file systems which manage data as a file hierarchy and block storage which manages data as blocks within sectors and tracks.[2] Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier

Минимальная информация про сервисы ostor on vstorage.

Os - хранятся данные в формате (uid, meta, data). Отдельный сервис os отвечает за свою область uid.

Ns - хранятся пути в лексикографическом порядке. Есть meta и свои uid

Из Ns есть ссылки в Os (объекту на os может соответствовать несколько путей). Эти ссылки консистентны. Также можно создать имя без объекта, наоборот нельзя

Есть агенты которые, если нужно, поднимают нужный сервис на текущей ноде, или наоборот его ложат. Также есть агенты, отвечающие за discover нод в кластере.

S3gw - обеспечивает gateway для доступа к ostor по amazon S3 rest api

Nfs/Pnfs.

Nfs протокол не масштабируемый - всё I/O проходит через mds

Pnfs позволяют клиенту получить layout (бывают разных типов : block /volume, object, file) и пока не истек его срок действия общатся с ds напрямую. Позволяет например в случае file layout делать striping и redudant(multipath). Механизм layout построен на существующих механизмах (то есть не на одном уровне с ними). Например метод layouget требует stateid open_state/shared_state/lock_state из nfs. Layout может отзываться по инициативе mds (это позволяет backchanel из nfs >= 4.1 версии). Коммитить можно и на ds и на mds. Можно (для одного клиента) отправлять параллельно несколько запросов на layout.

В nfs предусмотрено восстановление состояния после сбоя mds(grace period), клиента (специальные reclaim), ds (multipath).

Схема того, что есть.

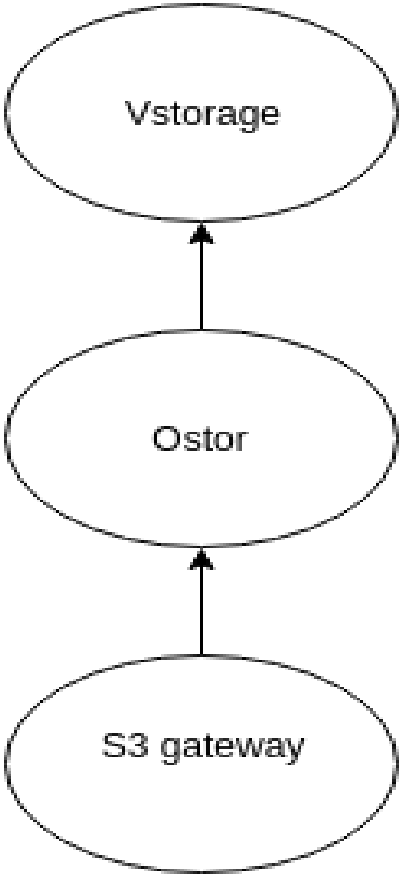
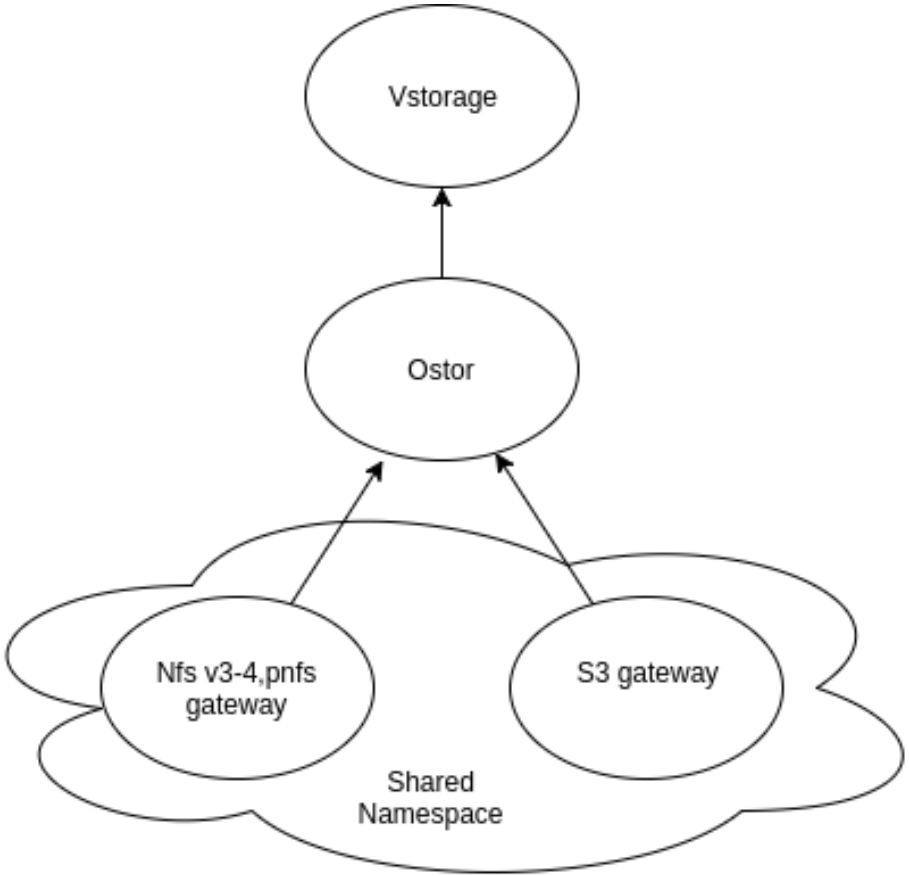


Схема того, что сделать.



Реализация.

Имплементация nfs - выбрана nfs-ganesha (сейчас наиболее стандартный вариант). Ганеша поддерживает механизм FSAL, который абстрагирует реализацию протокола от реализации нижлежащей файловой системы - если хотим поддержку nfs для своей фс, должны реализовать fsal_api функционал и собрать, как разделяемую библиотеку. Ганеша умеет например в runtime подгружать ту или иную библиотеку взависимости от типа фс. В частности, есть реализации FSAL для CEPH, GLUSTER, RGW, например.

Решено реализовать и вынести функции для работы с файлами (handle api) так же в отдельную библиотеку

Сейчас получаем такую схему.

Реализация fs_gateway.

Разделили на 3 части - поднятие фрейморка ostor client code для доступа, async и sync операций. Api ostor client code асинхронное, через него сделано async, через него sync (ожидаем ответа просто).

Апи на хэндлах (не дескрипторах) , имеющее posix-nfs подобный интерфейс.

Список sync вызовов, например

1) create_root, remove_root, get_root 2) create, mkdir [dir_handle, file_name, attrs] 3) unlink, rmdir [dir_handle, file_name] 4) lookup [dir_handle, file_name], walk [dir_handle, file_path] 5) lookup_ref [struct pcs_stor_obj_ref] (obtains handle without associated name 6) read, write [fh (maybe without associated name), offset, size] 7) rddir [handle] 8) setattr, getattr posix/extended [handle] 9) fsync [handle] 10) truncate [handle, size] 11) link [src_handle, dest_dist_handle, dest_file_name] (produces hard link here)

Этапы реализации fsal.

Fsal без pnfs - особо не трудно, почти 1 в 1 прокидываем sync вызовы fs_gateway

Pnfs fsal - ganesha поддерживает layout files, файл накрывается сегментом целиком. GetDeviceList нужны адреса ds-ов - немного дописываем код остор клиента, чтобы он предосталял uid -> adress. Сделано без multipath и страйпинга - почему - далее.

Переписка под ganesha extended api. Позволяет манипулировать состоянием, позволяет поддержать shared_reservations.

Рефакторинг получившегося инженером

Проблемы.

Ostor не поддерживает переименование дирректорий (без сумасшедшего оверхеда).

Решение (от инженеров) - глобальное изменение архитектуры ostor - дальше

Ostor не поддерживает нормально truncate и автоизменения размера объекта при записи (изначально ostor задумывался под s3 only - а там в основном нужен extend, да и перезапись в основном полная).

Решение (от меня) - написал патчи. Вот их commit mesag-ы:

OS : truncate support. Lazy block allocation support in obj_prepare_io. If operation is reading, than we will fill client buffer with zeros in obj_shedule_io. If operation is writing, than we will allocate nessesary extents/blocks. Extend support : with lazy block allocation we can just update size with PCS_STOR_OBJ_OPF_UPD_SIZE. Shrink support : removing unesessary extents/blocks with release_extent_data_blk and delete_extent_atomically (with preceding remove from btree). Clnt support : add flag PCS_STOR_OBJ_OPF_TRUNCATE, using which client makes clear, that he wants to reallocate blocks with size updating. Add update_size flag to api update operation params with same function.

Продолжение проблем.

OS : postwriting patch. Support writing big objects after obj creation with small size. If client wants to create auto resizieble object - he should put creating_object_is_autoresizeble flag to put_params. Autoresizeble object creates with PCS_STOR_ATTR_AUTO_RESIZABLE attribute. With each update_object calculates new size and if it's greater than old, it would update (flag PCS_STOR_OBJ_OPF_UPD_SIZE from client ignores). If client want's forcely update size, he should provides PCS_STOR_OBJ_OPF_TRUNCATE flag

В случае с pnfs не получится нормально сделать страйпинг и multipath - нужно прокидавание вызовов до vstorage - иначе получили просто, что записываем (на ds сейчас) объект, а его части (и он весь) могут лежать не на текущей ноде.

Решение (от инженеров) - глобальное изменение архитектуры ostor - дальше

При поддержке extended api ганеши код из понятного превратился из понятного в неочевидный и разросся раза в 3.

Нужны multiclient тесты на shared_reservations. Что сделано - в разделе про тестирование

Ещё проблемы.

При переходе на extended api отвалился некоторый функционал (потому что не реализован ещё). Например - нельзя делать делегации (кеширование на клиенте) из за такого куска

```
#if 0
/** todo FSF: re-enable delegation when I get more figured out. */
do_delegation(arg, res_OPEN4, data, owner, *file_state, clientid);
#endif
```

Написали мейнтейнерам - ждем ответа.

Были вроде бы ещё какие то проблемы

Тестирование.

Тестировали на 3 фреймворках.

Cthon-nfs-tests

Bfields-pynfs

Mora-nfstest. Используя его был написан мулти(2)клиент тест для проверки shared_reservations, о котором было написано ранее. Но, во-первых, фрейморк монтирует nfs, а линуксовое ядро не поддерживает shared_reservations, просто считается, что DENY_NONE, а ALLOW - в соответствии с режимом открытия файла. Нужны ещё тесты

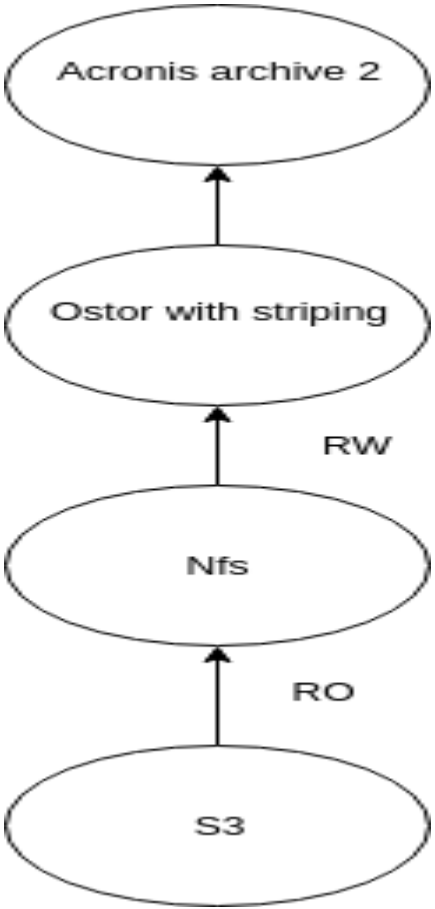
Что за глобальное изменение архитектуры.

Недавно были выкачены требования для ostor.

1) Redundancy per volume
2) De-duplication
3) Compression
4) Snapshot / RW snapshot
5) Shared NFS and S3 namespace
6) Directory rename in plain namespace
7) Reliability on corruption (1 PoF)
8) Seamless high availability
9) Encryption per volume
10) Tools
11) Quotas
12) Access from:

- NFS v3/v4, pNFS
- S3
- ISCSI
- FUSE ?

Предполагается сделать как то так.



S3-nfs sharing.

Как было показано, S3 предполагается над nfs(или fs_gateway) в режиме RO - предполагаемый юзкейс - быстрый листинг, через nfs (по юзкейсу) будут добавляться файлы. Юзерскоп разный для S3 и nfs.

Чем заниматься.

Юзерскоп и доступ в nfs. Например, для юзеркопа есть варианты через idmapper или kerberos. Есть вопрос насчет acl - были выпилены при рефакторинге, нужно добавить назад

Есть вариант сделать модуль fuse используя fs_gateway, замонтировать, затем расшарить самбой. Таким образом поддержать ещё и Samba gw.

Возможно - дизайн nfs + s3(rw). Это довольно трудная задача