

PNFS над объектным хранилищем Acronis

Александр Маркелов
Руководитель: Кирилл Коротаев
7 июня, 2019

Введение

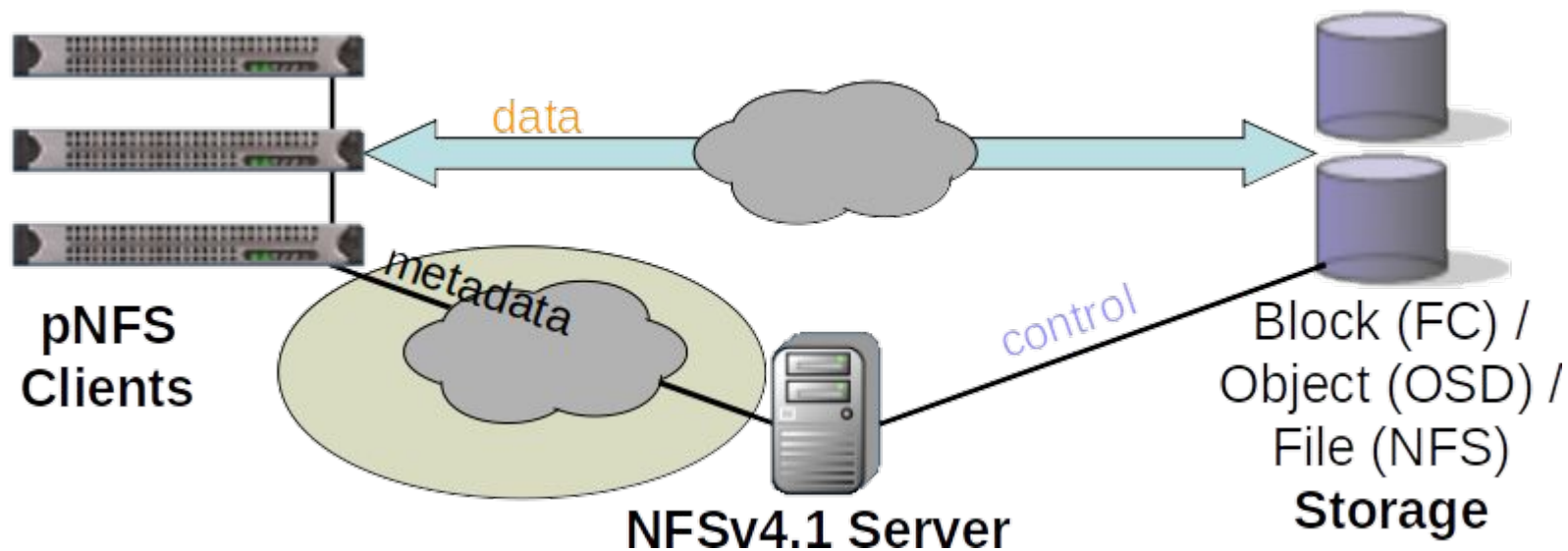
Nfs (Network file system)

- Основанный на RPC протокол, с вызовами к удаленной файловой системе: access, lookup, listing, open, close, read, write...

Pnfs (parallel nfs)

- Server обрабатывает lookup / listing / get запросы по метаданным
- Storage devices обрабатывают IO запросы по данным (взаимодействие с клиентом без участия server)
- Тип layout (files, block, objects) определяет протокол взаимодействия между клиентом и storage device

Pnfs



Acronis object storage

- NS (сервис имен): имена -> (uid, метаданные)
 - Отдельный NS отвечает за отдельный lexical или hash промежуток
- OS (сервис объектов): uid -> данные
 - Отдельный OS отвечает за отдельный промежуток по uid
- VCS(or pstorage) block (file) level backend
 - OS and NS хранят сущности в VCS
- Агенты для балансировки, обнаружения и обеспечения высокой доступности
 - Управляют отображением {os, ns} -> физические машины

Цель, требования, задачи

Цель

Получить реализацию pnfs над Acronis Object storage: то есть дать pnfs клиентам возможность работать с сущностями, хранящимися в Acronis object storage

Задачи

- Обозреть имеющиеся pnfs клиенты, определиться с поддерживаемыми типами layout для решения
- Обозреть имеющиеся реализации pnfs сервера и возможно выбрать определённую для адаптации
- Реализовать/адаптировать pnfs сервер над Acronis object storage
- Протестировать решение на функциональность и производительность

Требования

- Pnfs данные должны храниться в OS (но метаданные не обязательно должны хранится в NS)
- Решение может использовать интерфейс VCS, но не может изменять VCS

Решение

1. Поддерживаемые клиентами типы layout

- Linux kernel клиент: file и block layout.
- Red Hat Enterprise Linux клиент: file layout. Objects, blocks layout - beta
- CentOS 6.x клиенты: file layout (все типы - beta начиная с 7.x)
- Windows server (с 2012 версии) nfs клиенты: не поддерживают pnfs
- kofemann/ms-nfs41-client (для windows): file layout

Итого

- Хорошо поддерживается только file layout. Остальные типы - в нестабильных версиях
- Решение ДОЛЖНО поддерживать пока только file layout

2. Обзор реализаций pnfs серверов: nfs-ganesha (1):

- Userspace server, поддерживаются все типы layout
- Предоставляет File System Abstraction Layer (FSAL)
- Имеются реализации FSAL для GLUSTER, CEPH, ZFS, XFS, RGW и других

Nfs4j (2):

- userspace реализация на java, поддерживается file layout
- Предоставляет abstract filesystem backend (как nfs-ganesha)
- Менее удобна, чем nfs-ganesha

Linux kernel server (3):

Поддерживается только block layout

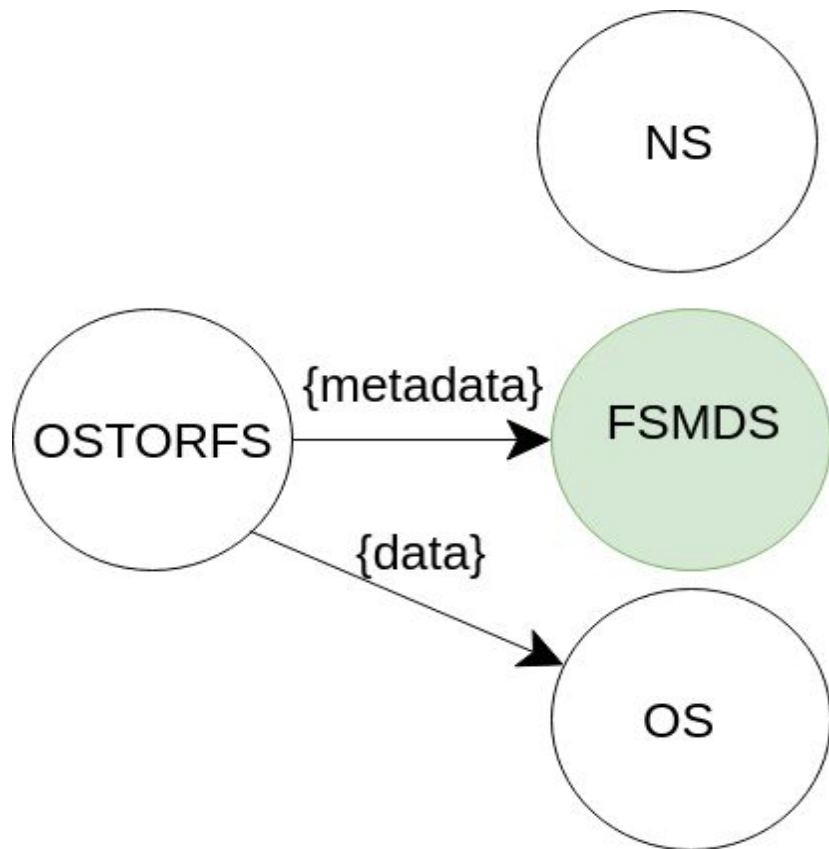
Итого

Рациональный выбор: реализация FSAL nfs-ganesha модуля для Acronis object storage

3. Дизайн FSAL модуля

- Вынесем file-level операции над acronis object storage в отдельную библиотеку ostorfs
- Ostorfs может быть полезна для реализации других fs-based протоколов над acronis object storage: например, WebHDFS
- FSAL модуль использует ostorfs
- Реализация FSAL интерфейса простая: почти вызовы из ostorfs

4. Дизайн ostorfs



NS не удобен для хранения метаданных pnfs

- Плохо без иерархии имен
 - RENAME(dir) проблемы
- Плохо с иерархией
 - Snapshots
 - Дорогие операции для s3 gateway, который уже использует NS

5. FSMDS: требования

- Поддержка иерархии имен
- Отображение {dir, name} -> {inode/dir_inode, symlink contents}
- Отображение {inode} -> {layout description, xattrs}
- Быстрый листинг содержимого директорий
- Поддержка коммитов
- Использование VCS для хранения всех сущностей

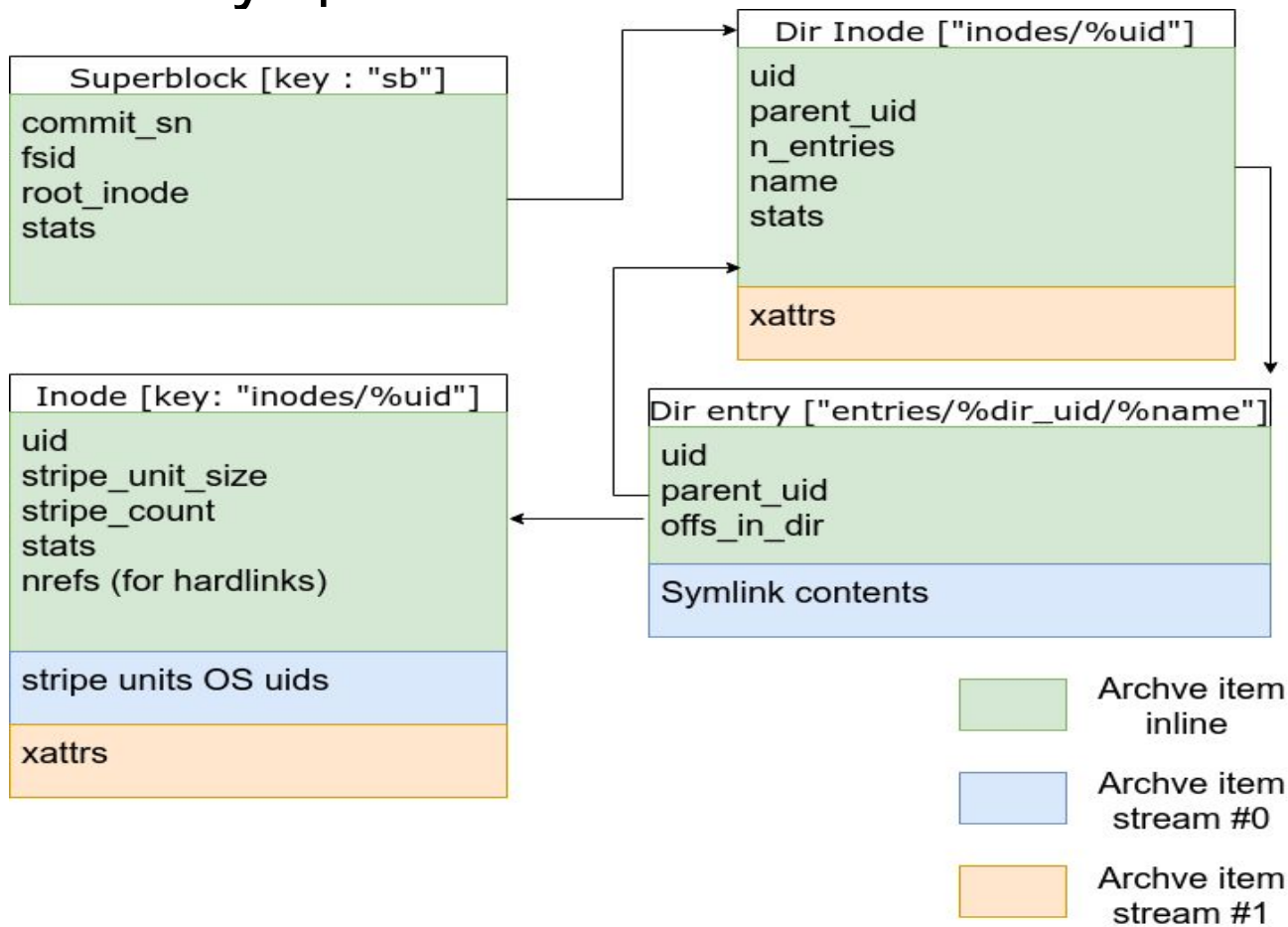
Есть ли библиотека, с помощью которой удобно реализовать FSMDS ?

Да, Acronis libarchive3

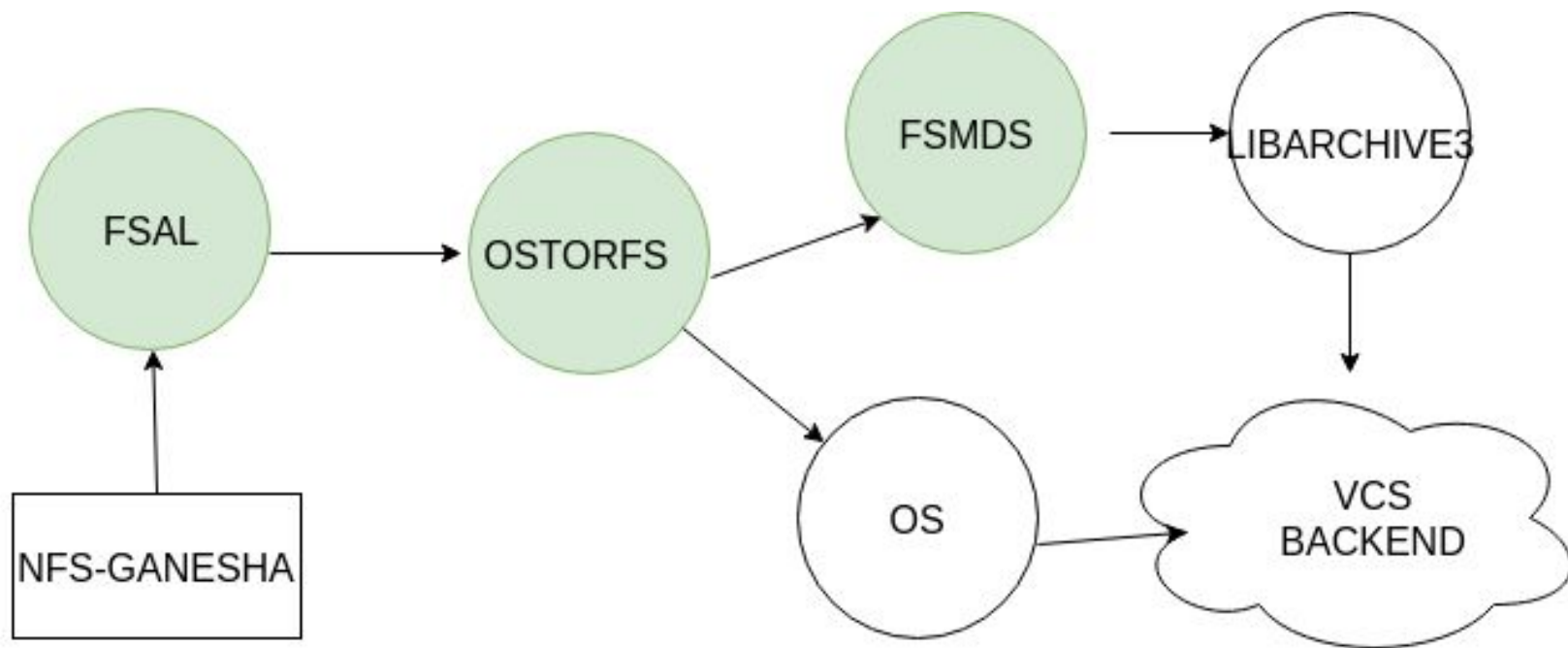
Acronis libarchive3

- Оптимизирован для хранения сущностей на VCS
- Основан на LSM деревьях
- Отображает Name -> archive_item
- Поддерживает хранение небольших данных ($\leq 1\text{KB}$) прямо в дереве (archive_item_set_uservalue)
- Поддерживает stream api для остальных данных (несколько stream для одного archive_item)
- Поддерживает коммиты

5. FSMDS: сущности



Финальный дизайн



Тестирование

1. Тестирование на функционал

- Требование: тесты, проходящие для nfs-ganesha FSAL с самым широким спектром возможностей (VFS FSAL), должны проходить и для ostorfs FSAL
- Требование проверено на тестирующих фреймворках:
 - dkruchinin/cthon-nfs-tests (nfs only)
 - Bfields-pynfs (nfs only)
 - Mora-nfstest (nfs + pnfs)

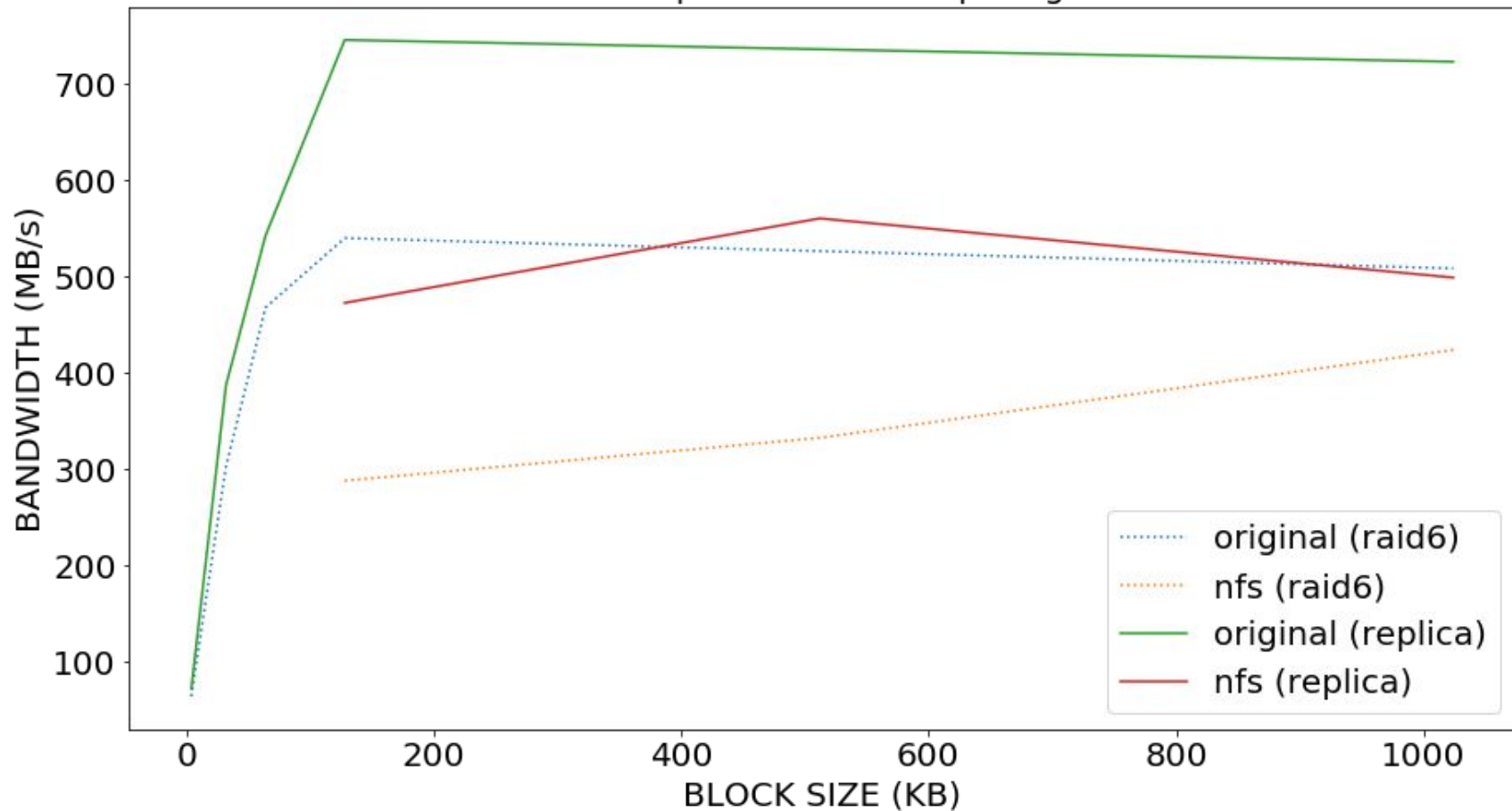
2. Тестирование производительности: конфигурация

- На каждой физической машине кластера запущен:
 - Один nfs-ganesha с duplex режимом:
 - Pnfs server для одного случайного FSMDS volume
 - Pnfs storage device для всех FSMDS volumes
 - Один FSMDS для уникального volume
 - Несколько OS сервисов
- Тестирование производилось на кластерах из 3 и 5 машин

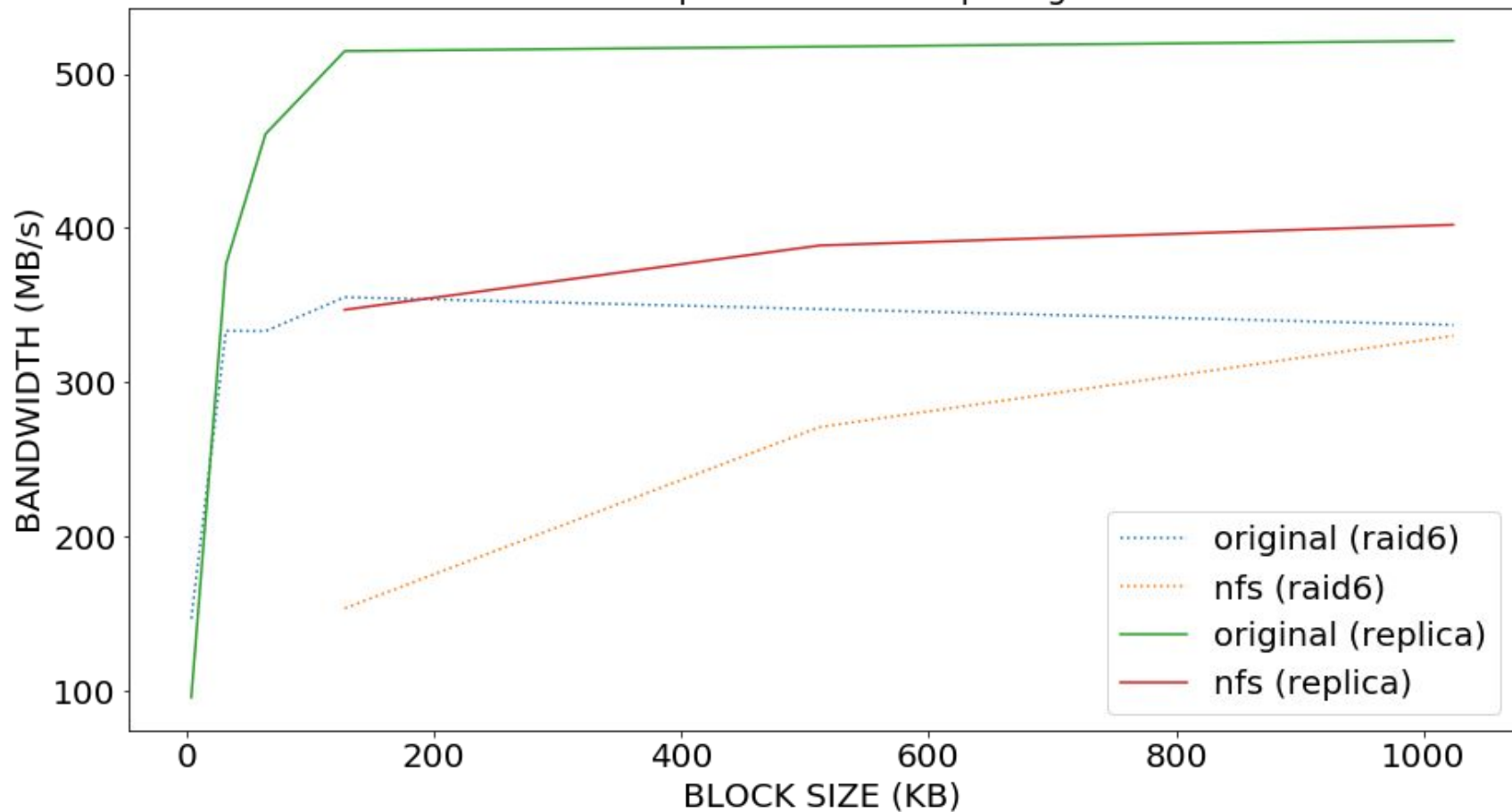
аппаратная конфигурация:

Storage	Processor	Memory
Cluster1		
2x480 GB SATA SSD 5x6 TB SATA HDD 5x3 TB SATA HDD	2xIntel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz	64 GB

Read performance comparing



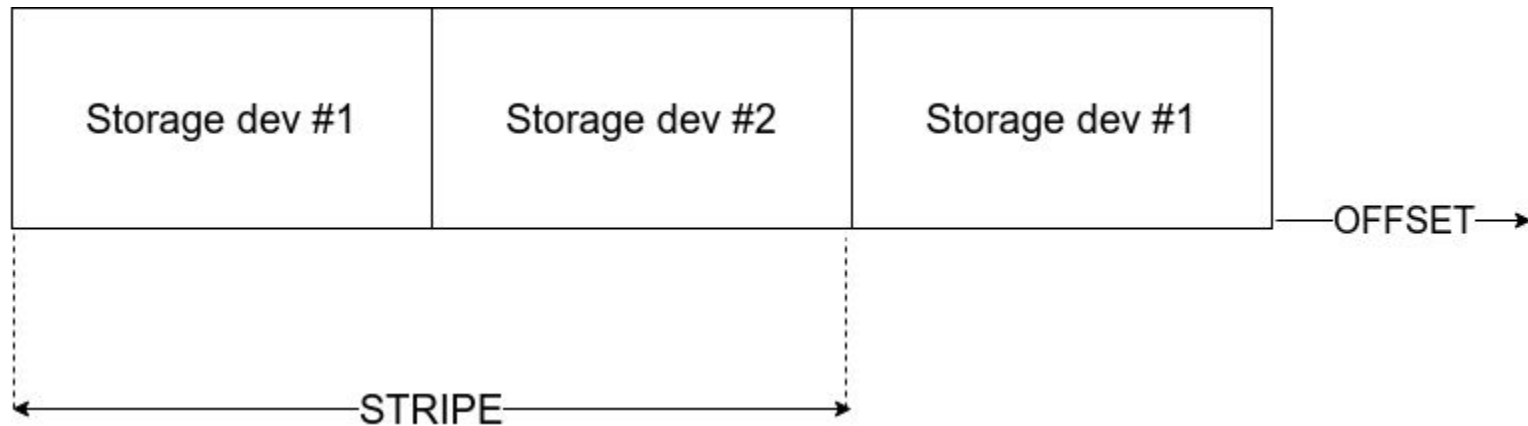
Write performance comparing



Итого

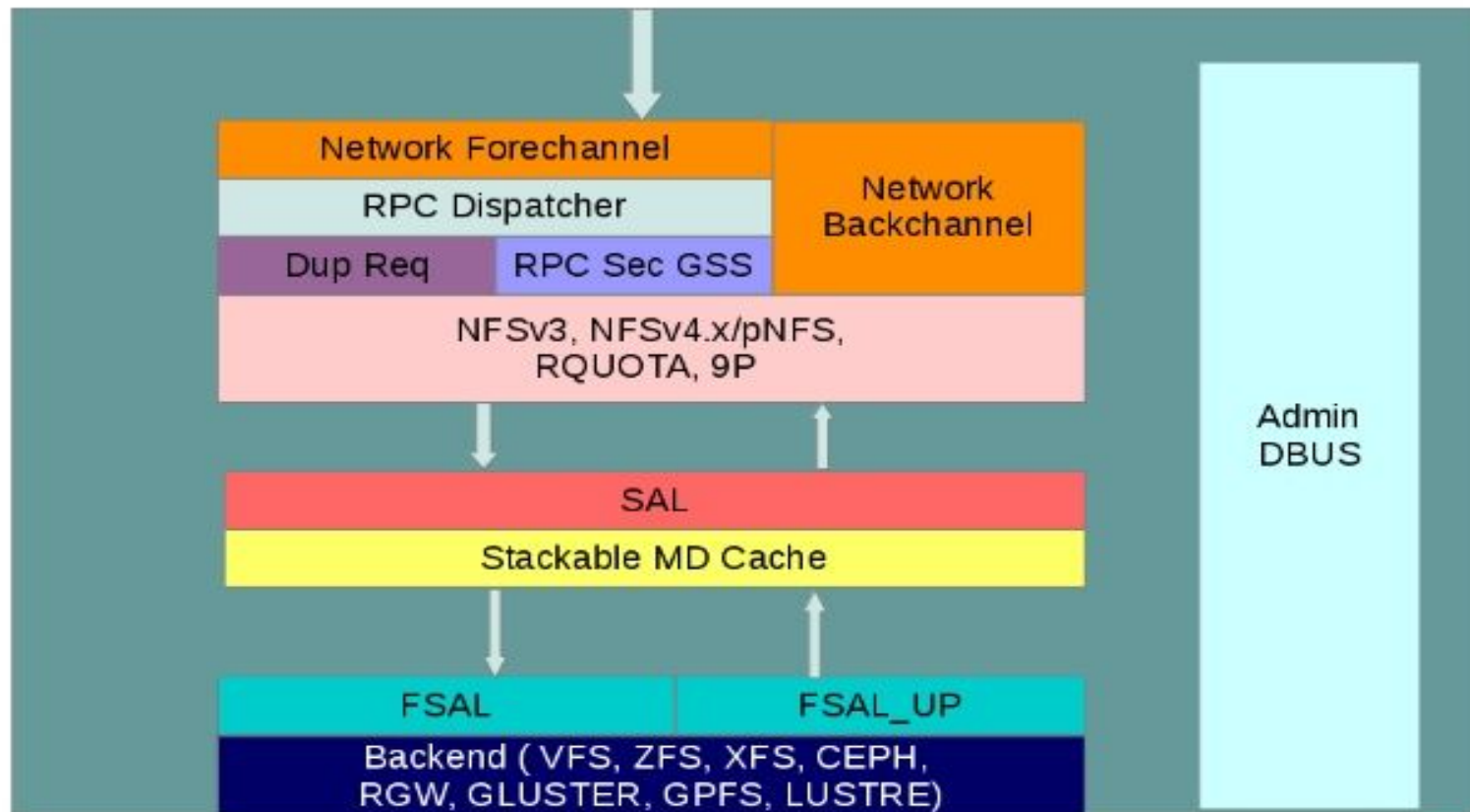
- Реализован fsmds object storage сервис
- Реализована ostorfs библиотека
- Реализован nfs-ganesha FSAL модуль для ostorfs
- Выполнено тестирование на производительность и функционал
 - Отставание от оригинальной производительности кластера в среднем ~1.3-1.5 раза, что примерно соответствует отставанию FSAL для других распределенных систем (CEPH, GLUSTER)

Pnfs: file layout



STRIPE COUNT = 2

NFS-Ganesha architecture



ostorfs sync interface

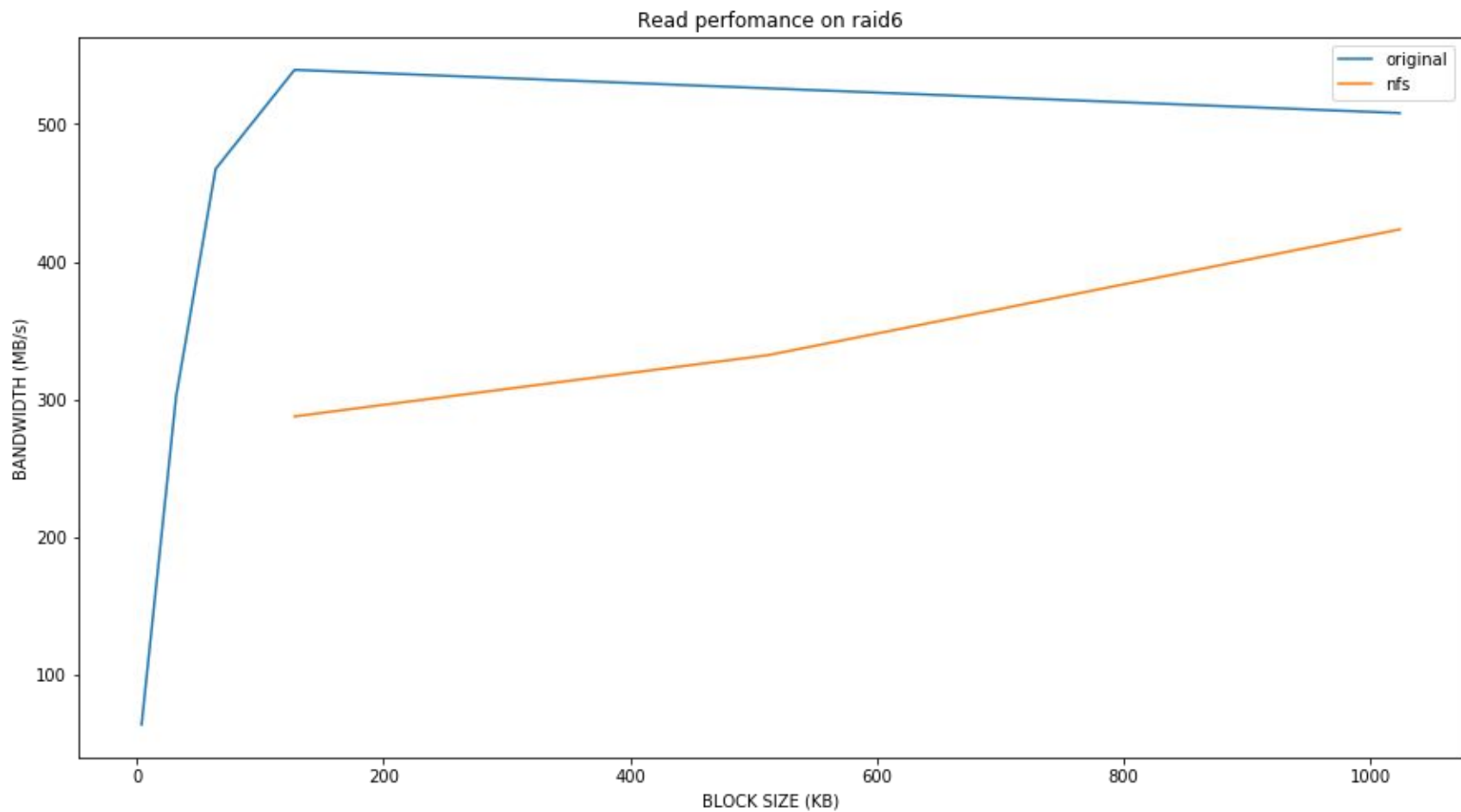
- CREATE(dir, name, stats)->obj, MKDIR(dir, name, stats)->obj
- UNLINK(dir, name, obj)
- OPEN(obj, flags), CLOSE(obj)
- GETROOT()->obj, LOOKUP(dir, name) -> obj, WALK...
- LINK(src_obj, dst_obj)
- SYMLINK(src_dir, src_name, dst_obj), READLINK(obj) -> (dir, name)
- RENAME
- TRUNCATE
- {GET/SET} ATTR (obj, stat); {GET/SET} XATTR (obj) by name, LISTXATTRS (obj),
{GET/SET} ACL (obj, acl)

ostorfs sync interface

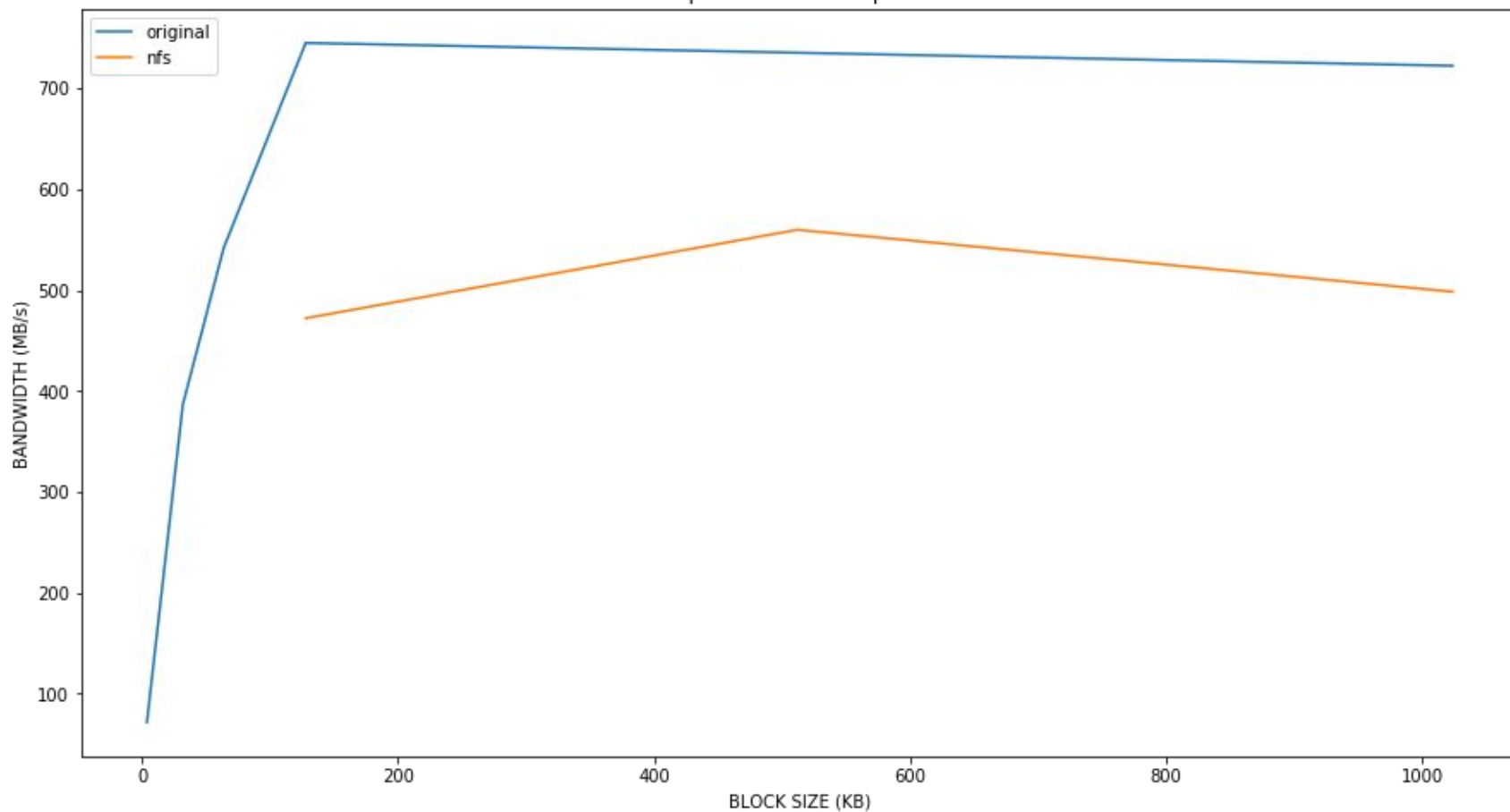
- OPENDIR, CLOSEDIR, READDIR->[]names(via iterator), SEEKDIR(off)
- LAYOUTGET(obj) -> []stripe_handle
- Separate stripe ios
 - READ_STRIPE(stripe_handle, off, buf), WRITE_STRIPE, FSYNC_STRIPE
- Common ios
 - READ(obj, offset, buffer), WRITE(obj, offset, buffer), FSYNC(obj, flags)
 - Parallel operations for object stripe units

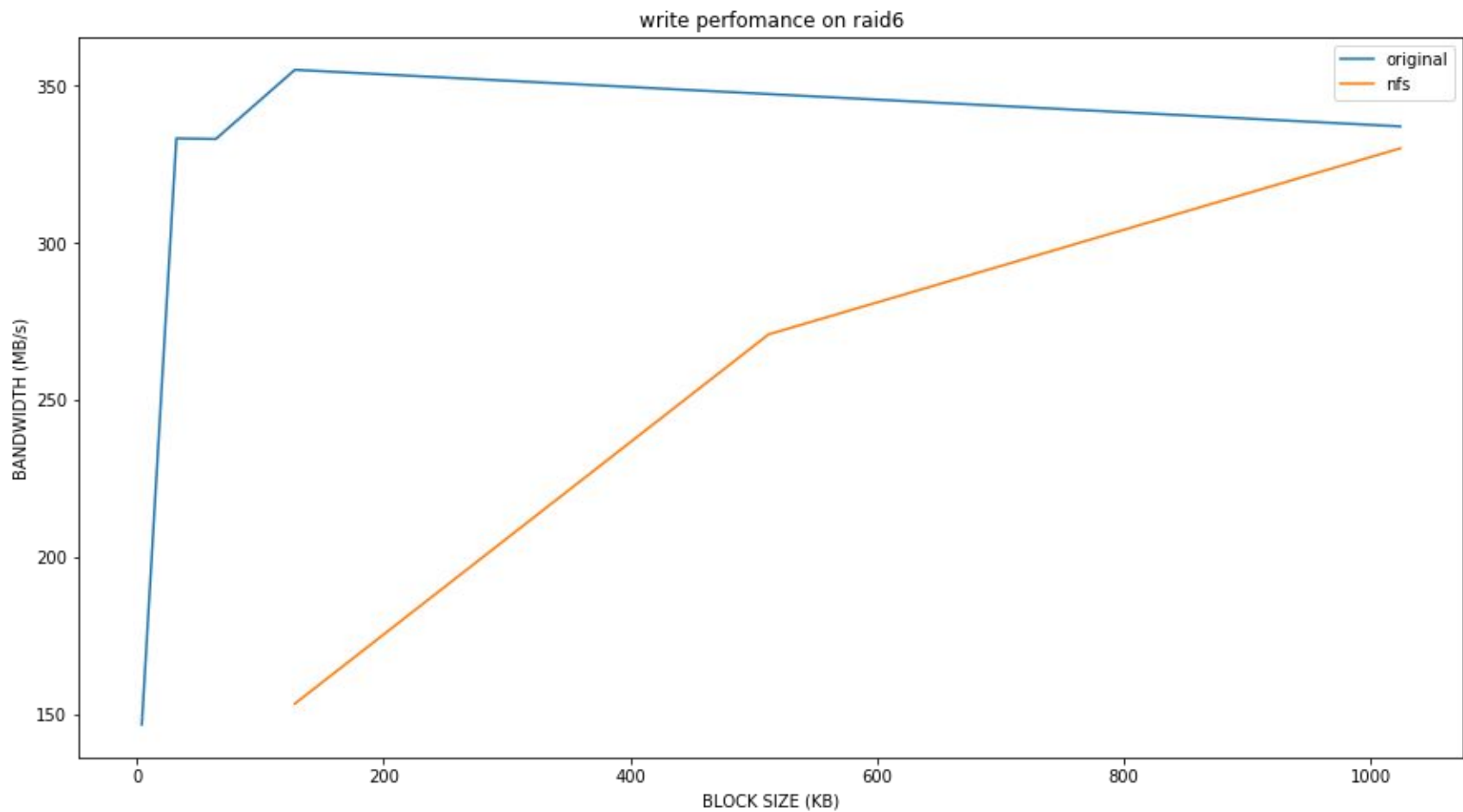
Fsmnds разное

- Inodes caches: RBtree (uid -> inode), LRU
- Отдельный archive для отдельного volume
- Журнал (write-ahead) для контроля операций, отправляемых из FSMDS к OS напрямую:
 - Записи: truncate_obj (truncate stripes) и delete_inode (delete stripes)
 - Запись хранится в archive_item
 - Журнал проигрывается до коммита и после чтения суперблока



Read performance on replica conf





write performance on replica conf

