Основы построения файловых систем

Сегодня мы поговорим про Copy-on-Write FS

ZFS (Zettabyte File System) – файловая система, изначально написанная для Sun Solaris.

WAFL (Write Anywhere File Layout) – файловая система, используемая в NetApp.

Они интересным образом объединяют RAID, менеджер томов и собственно ФС.

Требования к ФС и различные проблемы

1. ФС всегда должна быть в согласованном состоянии.
   - Требуется журналирование или идеи вроде LFS. В нагрузках, где часто меняются метаданные файлов, журнал удваивает количество требуемых записей.
   - Как загружаться с раздела, который не был «чисто» отмонтирован?

Требования к ФС и различные проблемы

1. ФС всегда должна быть в согласованном состоянии.
2. Быстрый FSCK или, что лучше, отсутствие оного.
   - Чтобы просто прочесть диск размером 10Tb требуется около суток. Такое время загрузки системы после сбоя неприемлимо.

Требования к ФС и различные проблемы

1. ФС всегда должна быть в согласованном состоянии.
2. Быстрый FSCK или, что лучше, отсутствие оного.
3. Быстрая запись в файлы и быстрая модификация метаданных.
   - В журналируемой ФС изменение метаданных подтверждается после записи в журнал. Эта запись линейная и делается достаточно быстро. Но при переполнении журнала скорость деградирует из-за необходимости модификаций разнесённых областей диска.

Требования к ФС и различные проблемы

1. ФС всегда должна быть в согласованном состоянии.
2. Быстрый FSCK или, что лучше, отсутствие оного.
3. Быстрая запись в файлы и быстрая модификация метаданных.
4. Гибкое управление размером ФС, возможность использовать несколько дисков одновременно для большей надёжности или скорости.
   - lvextend и resizefs не всегда позволяют изменить размер на лету (размер ext4 можно менять без отмонтирования только в Linux >= 3.3),
   - Добавление диска в RAID6 с помощью mdadm означает полное перестроение массива.

Требования к ФС и различные проблемы

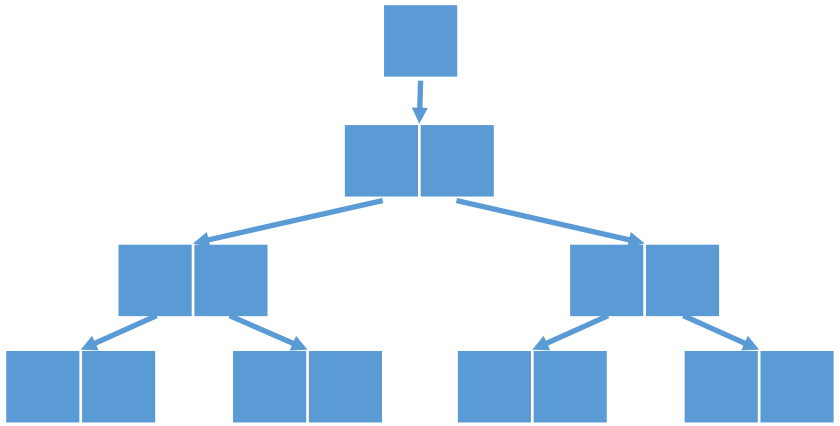
1. ФС всегда должна быть в согласованном состоянии.
2. Быстрый FSCK или, что лучше, отсутствие оного.
3. Быстрая запись в файлы и быстрая модификация метаданных.
4. Гибкое управление размером ФС, возможность использовать несколько дисков одновременно для большей надёжности или скорости.
5. Быстрые снимки состояния ФС, клоны ФС и откат к предыдущему состоянию.
   - Контейнеры,
   - Бекапы,
   - Безопасное обновление системы и откат неуспешных обновлений,
   - LVM COW snapshots при изменении одного из снимков раздела уменьшают производительность всех остальных,
   - LVM ничего не знает о том, что находится на разделе, поэтому снапшот части ФС сделать невозможно.

Требования к ФС и различные проблемы

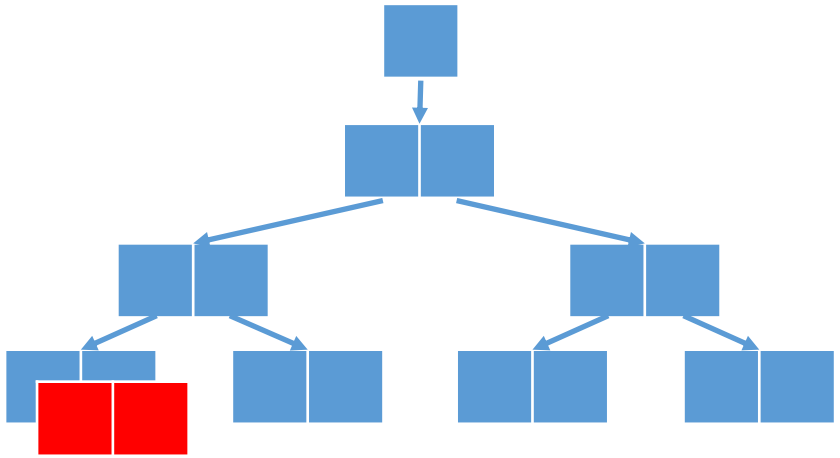
1. ФС всегда должна быть в согласованном состоянии.
2. Быстрый FSCK или, что лучше, отсутствие оного.
3. Быстрая запись в файлы и быстрая модификация метаданных.
4. Гибкое управление размером ФС, возможность использовать несколько дисков одновременно для большей надёжности или скорости.
5. Быстрые снимки состояния ФС, клоны ФС и откат к предыдущему состоянию.
6. Защита от случайных повреждений содержимого дисков. Для RAID, защита от RAID write hole.
   - Ext4 и XFS могут возвращать мусор при чтении (ср. эксперимент о порче содержимого дисков в CERN).

Идея: copy-on-write transactions (ZFS и WAFL)

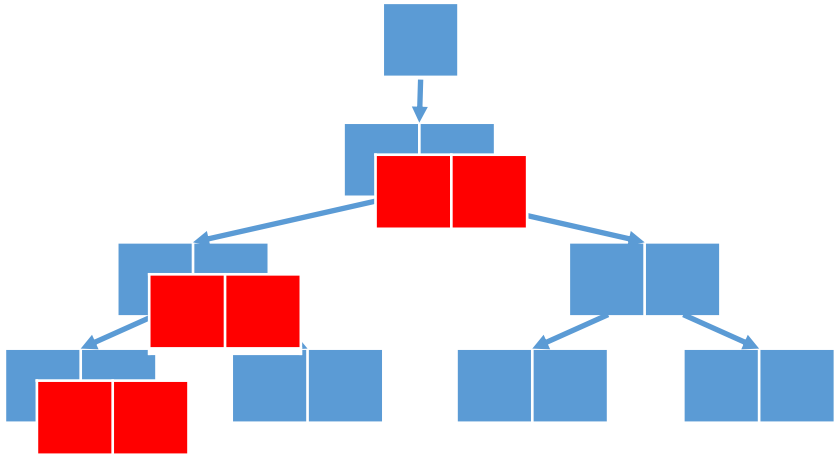
0. Исходное дерево



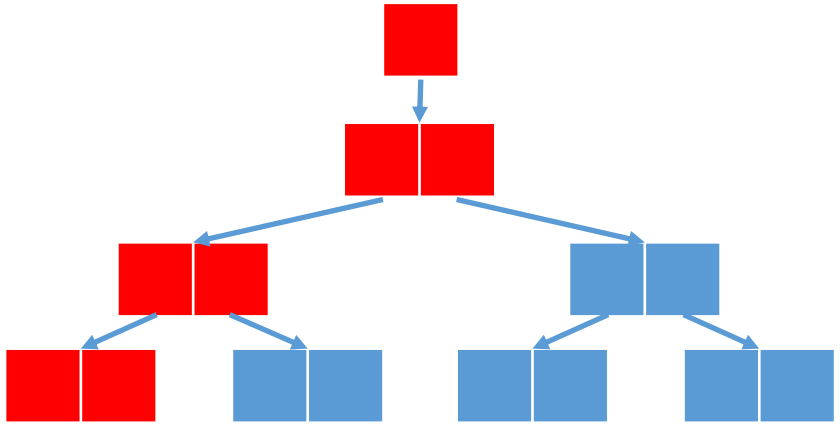
1. Создадим копии изменённых блоков



2. Создадим копии блоков выше по дереву



3. Перепишем суперблок ФС



Требования к ФС и что нам даёт Copy-on-write

ФС всегда должна быть в согласованном состоянии.	Суперблок всегда указывает на целостную ФС.
Быстрый FSCK или, что лучше, отсутствие оного.	FSCK не нужен.
Быстрая запись в файлы и быстрая модификация метаданных.	Блоки-копии можно выписывать последовательно, притом неважно, принадлежат они одному файлу или разным. **Вопрос:** как быть с фрагментацией файлов и производительностью чтения?
Гибкое управление размером ФС, возможность использовать несколько дисков одновременно для большей надёжности или скорости.	
Быстрые снимки состояния ФС, клоны ФС и откат к предыдущему состоянию.	Создание снимка бесплатно: надо просто не удалить старый суперблок, а сохранить ссылку на корень ФС как ссылку на корень снапшота. Откат к предыдущему состоянию тоже тривиален.
Защита от случайных повреждений содержимого дисков.	
Защита от RAID write hole.	RAID write hole возникают при перезаписи блоков, а в Copy-on-write FS перезаписи никогда не происходят.

Storage pool allocator в ZFS

ZFS реализует функциональность LVM внутри себя. Блоки выделяются из virtual devices (vdevs).

1. Vdevs организованы в дерево:
   - vdevs-листья – это просто диски,
   - составные vdevs – это набор подлежащих vdevs и правило преобразования данных при записи – mirroring, striping, Reed-Solomon,
2. место для объектов ФС выделяется из корневого vdev,
3. корневой vdev разбит на metaslabs (аналоги block groups в ext4),
4. внутри одного metaslab место разделено между несколькими slab allocators – аллокаторами, которые выделяют место блоками фиксированного размера; ZFS использует блоки размером от 512B до 128Kb.

Взаимодействие между SPA и вышележащими компонентами ZFS можно рассматривать как «SPA предоставляет malloc() для остальных компонент ФС».

Замечание: vdevs для RAID0 и RAID6 используют место эффективнее, чем RAID0 или RAID6, реализованные силами mdraid, поскольку могут выделять место блоками переменного размера. Если же ФС расположить на mdraid на RAID6-массиве с конфигурацией, например, 5+2, то блок ФС можно выбирать только кратным 5*4Kb.

Data management unit в ZFS

DMU предоставляет механизм хранения «файлов».

ZFS block pointer						
vdev0					grid	asize
G	offset0					
vdev1					grid	asize
G	offset1					
vdev2					grid	asize
G	offset2					
BDX	lvl	type	cksum	comp	psize	lsize
spare						
spare						
physical birth time						
logical birth time						
fill count						
checksum[0]						
checksum[1]						
checksum[2]						
checksum[3]						

- vdev – идентификатор vdev, на котором располагается блок,
- grid – информация о типе raidz (не используется),
- asize – размер блока с учётом заголовков ФС,
- offset – смещение внутри vdev,
- G – флаг “gang block”, блок, собранный SPA из нескольких меньших по размеру,
- B – флаг ”big endian”,
- D – флаг “deduped”,
- X – не используемый флаг,
- lvl – уровень косвенности данного указателя,
- type – тип объекта DMU (block pointer, master node, file data, ZAP, quota info, filesystem metadata, etc),
- cksum – тип контрольной суммы,
- psize и lsize – физический и логический размеры блока,
- physical birth time – номер транзакции, создавшей блок,
- logical birth time – номер транзакции, создавшей блок, для дедуплицированных блоков,
- fill count – количество ненулевых блоков, на которые ссылается этот указатель

Data management unit в ZFS

DMU предоставляет механизм хранения «файлов».

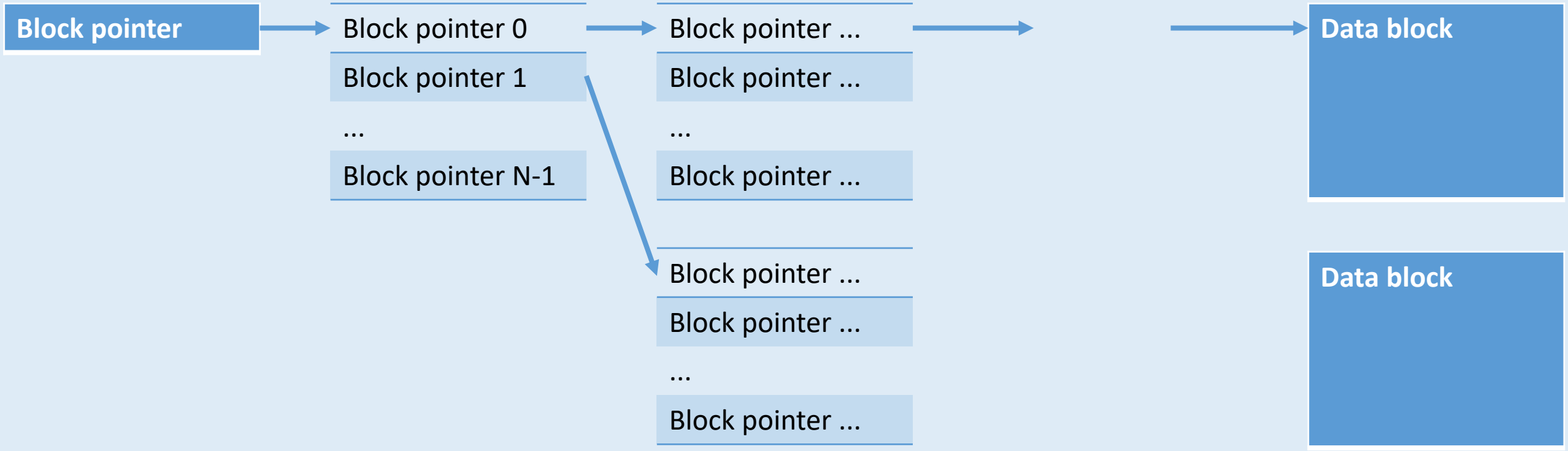
ZFS block pointer						
vdev0					grid	asize
G	offset0					
vdev1					grid	asize
G	offset1					
vdev2					grid	asize
G	offset2					
BDX	lvl	type	cksum	comp	psize	lsize
spare						
spare						
physical birth time						
logical birth time						
fill count						
checksum[0]						
checksum[1]						
checksum[2]						
checksum[3]						

Проверки целостности ФС:

- блоки могут хранится во многих копиях:
  - пользовательские данные обычно хранятся в одной копии,
  - метаданные ФС – в двух копиях,
  - метаданные пула – в трёх,
- контрольная сумма блока хранится отдельно от него – защита от одновременного повреждения блока и контрольной суммы,
- контрольная сумма вычисляется как криптографический хеш от данных,
- поле lvl избыточно и используется только для проверки структуры дерева блоков.

Data management unit в ZFS: ”объект” файловой системы

Один “объект файловой системы” или dnode, представляет собой дерево из указателей на блоки:



Замечание: в отличие от inode, содержащей direct block pointers, double indirect pointers и triple indirect pointers, dnode в ZFS имеет только один указатель на данные. Если блок данных имеет размер до 128Kb, то это будет прямой указатель на данные, иначе – указатель на дерево.

Data management unit в ZFS: файлы, каталоги и файловые системы

Файл в ZFS – это обыкновенный “объект” ФС.

Каталог – это ”объект” ФС, который трактуется как хеш-таблица, отображающая имя файла в его dnode.

Замечание о терминологии: хеш-таблицы в ZFS обрабатывает модуль ZAP, -- ZFS Attribute Processor, поэтому в документации можно встретить отсылки к “ZAP objects”.

Data management unit в ZFS: файлы, каталоги и файловые системы

Файл в ZFS – это обыкновенный “объект” ФС.

Каталог – это ”объект” ФС, который трактуется как хеш-таблица, отображающая имя файла в его dnode.

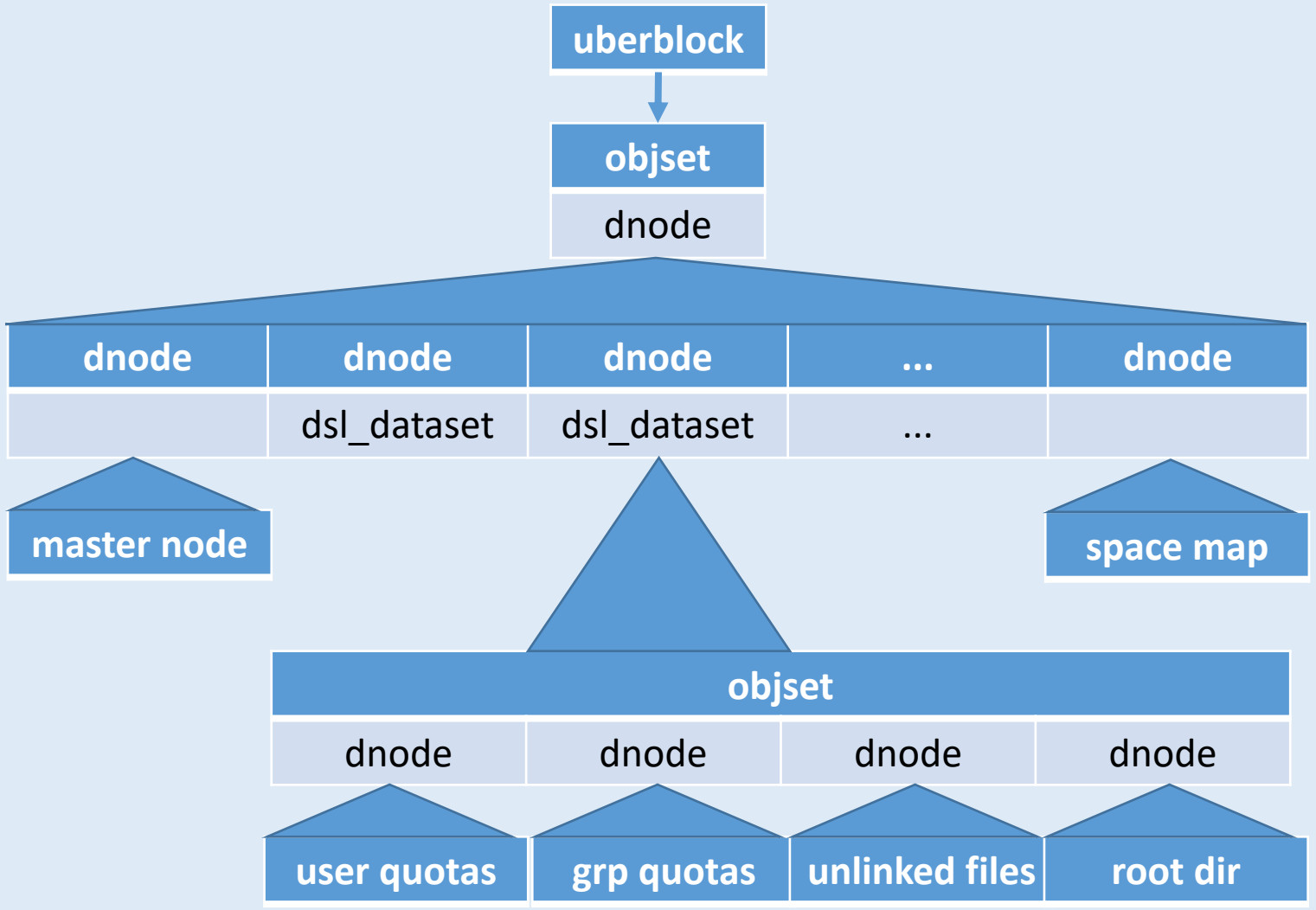
Замечание о терминологии: хеш-таблицы в ZFS обрабатывает модуль ZAP, -- ZFS Attribute Processor, поэтому в документации можно встретить отсылки к “ZAP objects”.

Файловая система в ZFS – это ”объект ФС”, который состоит из четырёх ссылок:

- ZAP object корневого каталога,
- ZAP object, отслеживающий квоты пользователей,
- ZAP object, отслеживающий квоты групп,
- ZAP object, отслеживающий открытые файлы, не имеющие имени.

Следствие: создавать FS, разделяющие ресурсы одного пула дисков, в ZFS так же просто, как создать каталог или файл.

Общий вид ZFS-пула



Checkpointing

Изменение одного блока требует переписать все вышележащие блоки.

Чтобы такая схема была эффективной, ZFS и WAFL накапливают достаточно много изменений в ФС перед тем, как выписывать их на диск.

Как реализовать fsync() на отдельных файлах без больших задержек?

- WAFL подтверждает операции с ФС после того, как сохранила их в журнале в NVRAM,
- ZFS подтверждает операции с ФС после того, как сохранила их в журнале ZFS Intent Log на лог-устройстве в пуле (обычно это SSD).

Требования к ФС и что нам даёт Copy-on-write

ФС всегда должна быть в согласованном состоянии.	Суперблок всегда указывает на целостную ФС.
Быстрый FSCK или, что лучше, отсутствие оного.	FSCK не нужен.
Быстрая запись в файлы и быстрая модификация метаданных.	Блоки-копии можно выписывать последовательно, притом неважно, принадлежат они одному файлу или разным.
Гибкое управление размером ФС, возможность использовать несколько дисков одновременно для большей надёжности или скорости.	SPA и ФС, которые являются файлами с точки зрения DMU.
Быстрые снимки состояния ФС, клоны ФС и откат к предыдущему состоянию.	Создание снимка бесплатно: надо просто не удалить старый суперблок, а сохранить ссылку на корень ФС как ссылку на корень снапшота.
Защита от случайных повреждений содержимого дисков.	Криптографические хеши в качестве контрольных сумм, организация всех данных пула в виде дерева Меркле.
Защита от RAID write hole.	RAID write hole возникают при перезаписи блоков, а в Copy-on-write FS перезаписи никогда не происходят.

Минусы COW

- Переписывание деревьев имеет хорошую **амортизированную** сложность, но дорого, если изменений в ФС происходит мало.
- Нетривиальная процедура чекпоинтинга (вопрос: как ядро обрабатывает изменения в ФС, происходящие во время чекпоинтинга?),
- Требуется отдельный лог, чтобы не допускать больших задержек на маленьких транзакциях,
- Любое изменение ФС требует наличия свободного места.