Восстановление ФС. Журнальные ФС

Д.В. Иртегов

Курс "Операционные системы"

ФИТ/ФФ НГУ

Целостность метаданных

- Список (битмап) свободных блоков и списки блоков файла находятся в разных местах
  - Их нельзя обновить атомарно
  - Могут получаться потерянные блоки или блоки, посчитанные дважды
- Выключение электричества на ваши блокировки не смотрит
  - Поэтому программные и даже аппаратные блокировки бесполезны
- Аналогичная проблема существует в базах данных

Что делать?

- Три основных подхода:
  - Fsck/chkdsk
  - Журнальные ФС
  - Copy-on-write (log structured) ФС

fsck/CHKDSK: dirty flag

- Завести в суперблоке ФС флаг загрязнения (dirty flag)
- Устанавливать при монтировании ФС на чтение-запись
- Можно не устанавливать при монтировании только на чтение
- При размонтировании, сбрасывать
  - разумеется после записи всех кэшей
- Если при монтировании ФС флаг установлен, значит, данные на диске могут быть не целостными
- Надо чинить.

fsck/CHKDSK: как чинить?

- Просматриваем все иноды, строим список занятых блоков
  - Похоже на mark’n’sweep
- Все остальные блоки объявляем свободными
  - Выкидываем старый список свободных блоков
    и заменяем на то, что получилось
- Просматриваем все каталоги, строим список инодов с ненулевым количеством хардлинков
  - Обновляем счетчики хардлинков
  - Иноды с ненулевым счетчиком, но без ссылок (сироты)
    складываем в lost+found

Практические сложности

- Для больших дисков и ФС где много файлов это очень долго
- Что делать, если промежуточные данные не влезают в память?
- Как восстанавливать корневую ФС (диск C: в Windows)?

Журнальные ФС

- Вводим понятие транзакции
- Транзакция – это группа операций, которая либо выполняется вся целиком, либо не выполняется совсем
- Целостность данных может нарушаться только внутри транзакции
- Разработчики ОС не знают, каковы критерии целостности для пользовательских данных, поэтому обычно в транзакции включают только метаданные ФС
- Как обеспечить атомарность транзакции при выключении электричества?

Журнал (log)

- На диске (не обязательно на том же) создается линейное хранилище (журнал)
- Все транзакции выполняются в два этапа
- Сначала описание транзакции пишется в журнал
- В конец описания добавляется запись commit (конец транзакции)
- Дожидаемся, пока описание транзакции физически ляжет на диск
- Начинаем накатывать изменения на саму файловую систему
- Создаем в журнале запись SYNC
  (указывает, какие изменения уже внесены в ФС)

Восстановление журнала

- Три возможных сценария:
- Последняя запись в журнале – SYNC
  - Все транзакции накачены на диск, данные целостны, чинить нечего.
- Последняя запись в журнале незаконченная (нет ни SYNC, ни COMMIT)
  - Эта транзакция на диск не накатывалась, данные целостны, чинить нечего
- Последняя запись содержит COMMIT, но не SYNC
  - Надо доделать или откатить эту транзакцию

Преимущества журнальных ФС

- Фиксированное (и, обычно, небольшое) время восстановления после сбоя, независимо от объема диска
- Можно агрессивнее кэшировать данные при записи
- Можно записывать данные в фоновом режиме, когда дисковый контроллер не занят другой работой

Недостатки журнальных ФС

- Данные пишутся на диск два раза
  - Только метаданные
  - Операций записи вообще довольно мало
  - Обычно компенсируется более агрессивным кэшированием
- Код драйвера ФС усложняется => выше вероятность ошибок
- Соблазняет разработчиков создавать более сложные ФС
- Сценарий потери данных, не имеющий аналогов в традиционных ФС:
  - Из-за ошибки в драйвере, в журнал записываются некорректные данные
  - Данные на диске корректны
  - Такая ошибка обнаруживается *только* если сбой происходит в момент, когда в журнале есть некорректная запись

Copy-on-write (log structured) ФС

- С идеей copy-on-write мы знакомились в ISO 9660 (CDFS)
- Все новые данные всегда пишутся на новое место
- + Старые данные остаются целостными
- - При каждой записи надо обновлять метаданные
- Метаданные обновляются рекурсивно вплоть до корневого объекта: т.наз. корневой инод (root inode)
- Обновления объединяются в пакеты, накат каждого пакета приводит к созданию новой копии корневого инода
- Каждая копия корневого инода – это снимок состояния ФС на некоторый момент времени

Проблемы такого подхода

- Что делать с данными пакета, накат которого еще не завершен?
- Что делать, когда места под корневой инод кончатся?
- Как управлять свободным пространством?

Что делать с данными незавершенного пакета?

- Журнал
- Журнал не такой, как в собственно журнальных ФС
- Туда пишутся все данные, записываемые в ФС
- Журнал разбит на секции, накат каждой секции заканчивается созданием нового снимка (записью корневого инода)
- При восстановлении ФС достаточно накатить незаконченные секции журнала

Сколько может быть корневых инодов

- Понятно, что их количество ограничено
  - NetAPP WAFL: 256
  - ZFS: 64k
- Старые корневые иноды (старые снимки) удаляются
- Можно руками помечать некоторые снимки как неудаляемые
- Можно делать снимки и клоны каталогов и отдельных файлов

Как управлять свободным пространством?

- CoW ФС не хранят на диске карты свободного пространства.
- Например, NetAPP WAFL хранит в каждом снимке битмап блоков, занятых этим снимком.
- Битмапы тоже хранятся по принципу CoW,
  т.е. в снимке хранятся только отличия от предыдущего снимка
- Драйвер ФС в памяти строит OR всех битмапов
  и получает точную карту свободных блоков на диске
  - Нужно много памяти
  - Медленное монтирование
  - Удаление снимка – дорогая операция
    (в худшем случае надо перестроить всю карту)

Применение CoW ФС

- NetAPP WAFL, Sun ZFS, Linux btrfs
- Системы хранения данных (NetAPP, Nexenta)
- Снимки и клоны дисков iSCSI, образов виртуальных машин и контейнеров
- В Solaris 11, при установке апдейтов ОС создается снимок (bootenv), и вы можете загрузить старую версию ОС