ПОДДЕРЖКА ВЕРСИЙ ФАЙЛОВОЙ СИСТЕМЫ В МЕНЕДЖЕРЕ ЛОГИЧЕСКИХ ТОМОВ LINUX

А.С. Игумнов

Введение

Системы хранения версий файлов хорошо известны разработчикам программного обеспечения. Они позволяют вносить изменения в код программы без риска потери ранее найденных удачных решений, отменять неудачные версии, сравнивать код различных версий. Версии хранятся в виде правок к предыдущей версии, что экономит дисковое пространство.

При администрировании вычислительных серверов иногда возникают ситуации, в которых желательно было бы, чтобы вся файловая система (ФС) вела себя как система хранения версий.

К таким ситуациям можно отнести:

1. восстановление состояния ФС после неосторожных действий пользователя;
2. поиск изменений произведенных хакерами при взломе системы;
3. сохранение состояния ФС одновременно с сохранением контрольной точки вычислительной задачи;
4. резервное копирование файлов без остановки программ.

Первые три задачи решаются резервным копированием файлов, но требуется много времени как на сохранение данных, так и на их восстановление.

Четвертая задача решается модулем dm_snapshot, входящим в стандартную поставку Linux, но при этом скорость доступа к ФС падает в два раза.

В данной статье предлагаются алгоритмы и структуры данных, которые позволяют реализовать поддержку версий ФС средствами менеджера логических томов Linux. Эти алгоритмы не зависят от типа используемой ФС и не требуют внесения правок в драйвер ФС.

Используемые сокращения:

ФС - файловая система;

LVM - Logical Volume Manager, менеджер логических томов;

DM - Device Mapper;

PV - Physical volume, физический том;

VG - Volume group, группа томов;

LV - Logical volume, логический том;

PE - Physical extent, физический экстент;

LE - Logical extent, логический экстент.

Принципы работы менеджера логических томов

Менеджер логических томов (LVM) [1] представляет собой двухуровневую систему косвенной адресации блоков данных на дисковых устройствах. На нижнем уровне расположены физические тома (PV): например жесткие диски или дисковые разделы. Несколько физических томов для создания единой непрерывной нумерации блоков данных объединяются в группу томов (VG). На верхнем уровне находятся логические тома (LV). Адрес блока в LV по определенному алгоритму отображается на адрес блока в VG, который, в свою очередь, отображается на адрес блока одного из PV входящих в VG. Данные операции могут производиться рекурсивно: один или несколько LV могут быть объединены в VG, на которой, в свою очередь, могут быть созданы новые LV.

На уровне VG помимо линейного отображения можно программно организовать RAID различных уровней, а также горячую замену дисков с прозрачным для LV переносом данных. На уровне LV реализуется изменение размера логических томов, шифрование данных, "замораживание" (snapshot) текущего состояния тома.

Драйверы файловых систем в Linux оперируют только с LV, что позволяет выполнять операции уровня VG в режиме online. Некоторые операции на уровне LV могут проводиться в режиме offline, например создание snapshot'ов, другие требуют приостановки доступа к логическому тому, например изменение размера тома.

На рис.1 изображена иерархия в LVM.

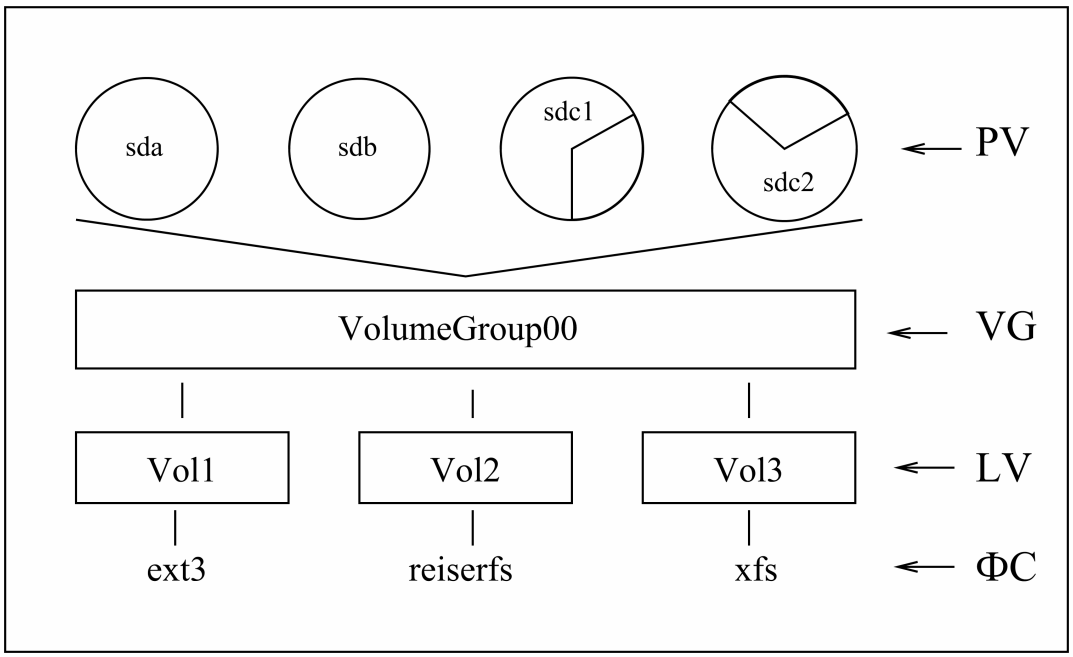

Рис.1

Для оптимизации по памяти блоки данных, называемые в LVM экстентами, имеют достаточно большой размер (4-32 Мбайт). Термин "логический экстент" (LE, Logical extent) применяется для описания блока данных на LV, а "физический экстент" (PE, Physical extent) для описания блока данных на PV. Размер LE определяется на уровне группы томов, а размер PE совпадает с размером LE для всех физических томов в группе.

Реализация LVM в ядре Linux базируется на подсистеме Device Mapper (DM) [2]. В подсистему DM входят драйверы, реализующие как VG, так и LV. Каждый драйвер регистрирует в ядре интерфейс блочного устройства, извлекает из очереди ввода/вывода запросы для данного устройства, переписывает адрес назначения запроса и снова помещает запросы в очередь на обработку. Таким образом запрос обрабатывается цепочкой драйверов, пока в адрес назначения не будет вписан физический том.

Каждый запрос ввода/вывода содержит:

– тип операции (чтение или запись);
– блочное устройство, для которого предназначен запрос;
– смещение в экстентах относительно начала устройства;
– размер буфера, кратный размеру экстента;
– указатель на буфер с данными.

При обработке запроса могут быть изменены любые из этих полей. Блочное устройство заменяется на нижележащее при записи или на вышележащее при чтении; смещение пересчитывается в соответствии с алгоритмом отображения; данные могут шифроваться при записи и дешифроваться при чтении.

Драйвер может регистрировать несколько блочных устройств для различного представления одних и тех же данных. Например, драйвер dm_snapshot предоставляет два блочных устройства: текущее состояние тома и его "снимок" - состояние на некоторый момент времени.

Требования к драйверу поддержки версий LV (dm_version)

На основе списка задач, требующих сохранения версий ФС, были сформулированы следующие требования к реализации драйвера:

- Создание версий ФС производится без поддержки драйвера ФС средствами DM;
- По возможности новые версии создаются в режиме online;
- Создание новой версии должно происходить максимально быстро;
- Запись экстента в текущую версию, не должна приводить к операциям чтения или записи в предыдущих версиях;
- Экстент, который не менялся на протяжении нескольких версий, должен храниться на диске в одном экземпляре;
- Чтение сохраненных версий возможно одновременно с полным доступом к текущей версии;
- Возможен откат текущей версии до любой из сохраненных;
- При сохранении максимально доступного числа версий, наиболее старая из сохраненных версий автоматически удаляется, а освободившееся место используется для сохранения новой версии;

- В случае программного или аппаратного сбоя сохраненные версии не должны быть утеряны.

Объем LV, доступный в текущей версии, предполагается равным исходному дисковому пространству, деленному на количество хранимых версий. Это ограничение вводится с учетом того, что все экстенты в каждой версии могут быть перезаписаны. Соответственно, необходимо резервировать пространство для хранения полной копии каждой из версий.

Основные структуры данных

LV с поддержкой версий хранится в VG в виде блока со служебной информацией и слотов хранения данных. Все слоты хранения данных имеют равный размер, определяющий объем, доступный для хранения одной версии LV. Количество слотов задается при инициализации LV и ограничивает количество хранимых версий. LE, находящийся по смещению X от начала LV, отображается в PE, находящийся по смещению X от начала одного из слотов.

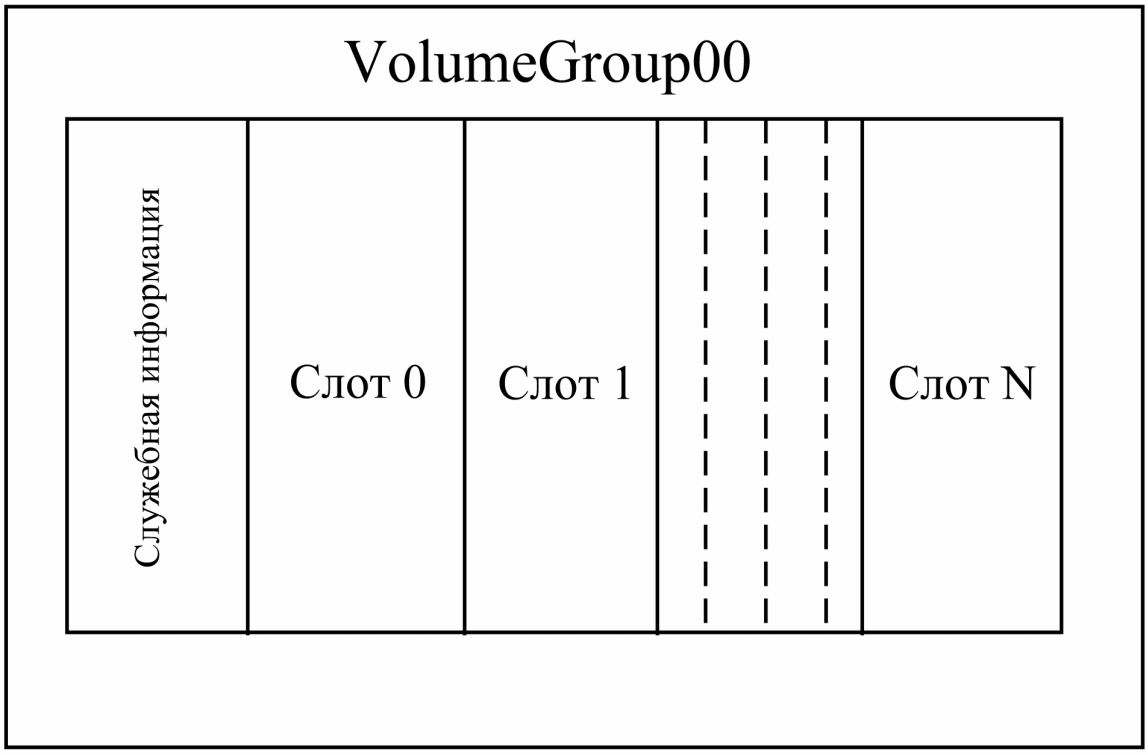

Рис. 2

На рис.2 показано размещение структур LV внутри VG.

Служебная информация содержит:

- число хранимых версий num_versions;
- последнюю сохраненную версию current_version;
- самую раннюю из сохраненных версий oldest_version;
- размер экстента chunk_size;
- количество экстентов в одном слоте хранения данных slot_size;
- массив структур vchunk размерности slot_size.

Структура vchunk содержит информацию о соответствии номеров версий LV и номеров слотов, использованных для хранения конкретного LE.

Номера слотов хранения данных нельзя однозначно сопоставить с номерами версий LV. Прямое отображение номеров привело бы к необходимости копировать все PE в новый слот при каждом переходе на новую версию. Было принято решение хранить историю записей в каждый LE и выделять PE в новом слоте только при условии, что в предыдущей версии была запись в старый слот. На рис. 3 показано распределение PE по слотам после сохранения третьей версии. На данной схеме LE[0] и LE[1] были изменены только в версии 0 или вообще не менялись; LE[3] изменялся в версиях 0 и 1; LE[1] менялся во всех трех версиях.На рис. 3- Отображение LE на PE

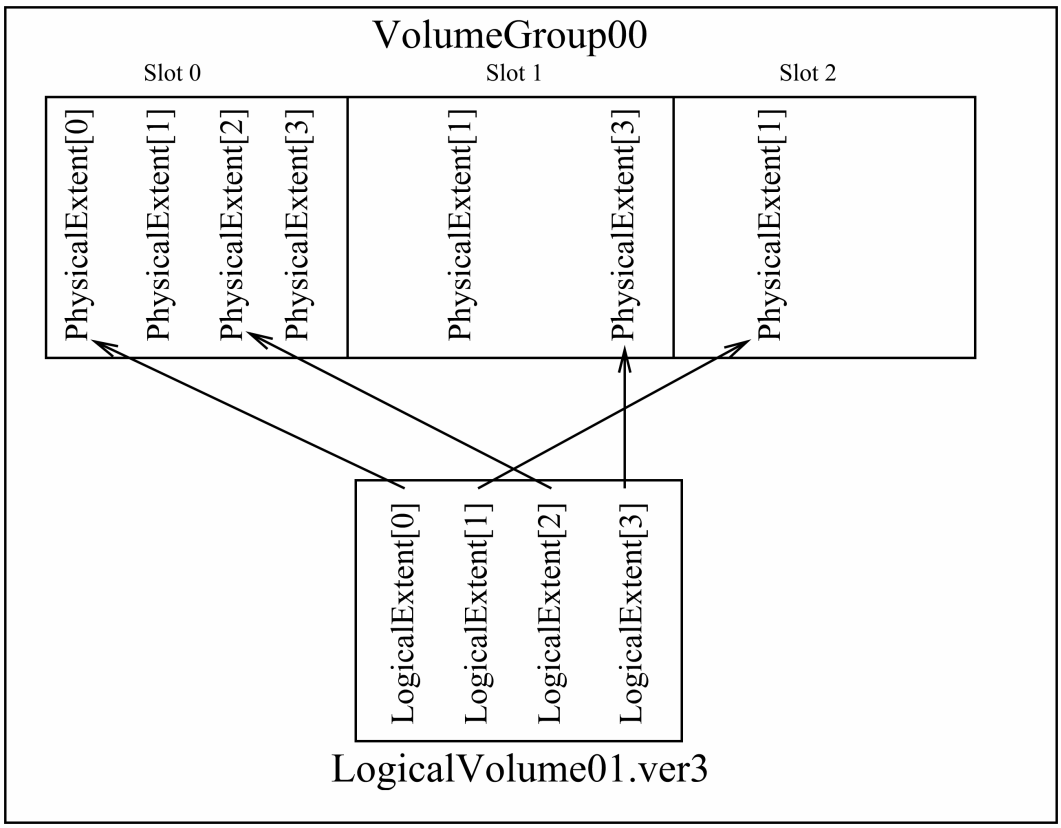

Рис. 3

Структура vchunk, хранящая историю записей в конкретный LE, состоит из двух полей:

- номер слота, в который производится запись, write_slot;
- битовая карта версий write_bitmap.

В write_bitmap каждой сохраненной версии LV соответствует один разряд. Если он равен единице, то в данный LE в соответствующей версии была запись, если он равен нулю, то записи не было. На рис. 4 показана ситуация, в которой запись в LE происходила только в версиях 0 и 2.

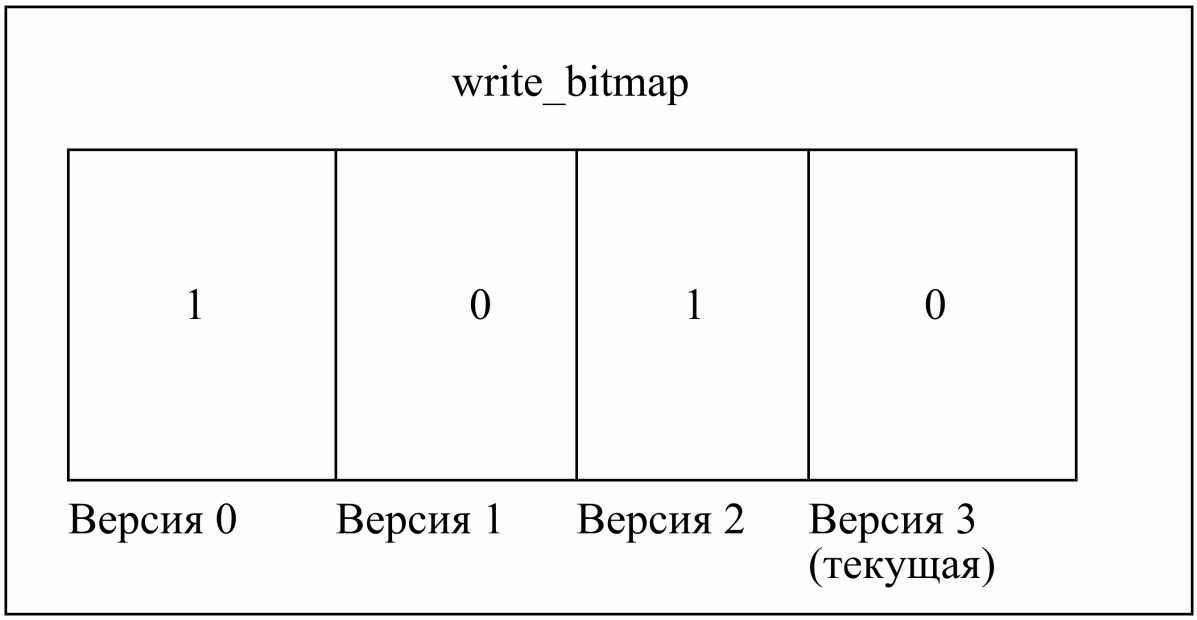

Рис. 4

Вычисление номера слота при операциях чтении/записи

Чтение данных возможно как из текущей версии LV, так и из сохраненной. Запись данных возможна в любой экстент текущей версии LV и в те экстенты сохраненной версии LV, которые были изменены в данной версии.

Правила вычисления слота для записи:

- если запись идет в текущую версию, то она производится в слот определяемый полем write_slot;
- если запись идет в сохраненную версию, write_slot равен write_slot минус количество тех единиц в write_bitmap, которые "новее" нужной версии;
- в момент записи write_bitmap[current_version] устанавливается в 1.

При переходе на новую версию:

1. current_version увеличивается на 1;
2. write_bitmap[current_version] устанавливается в 0;
3. проверяется write_bitmap[current_version-1] и, если он был равен 1, write_slot увеличивается на 1;
4. если write_slot получился равным num_versions, то write_slot сбрасывается в 0.

В приведенном на рис.4 примере write_slot должен быть равен двум и останется равным двум при переходе на четвертую версию.

Правила вычисления слота для чтения:

- если данные читаются из текущей версии, read_slot равен write_slot;
- если данные читаются из сохраненной версии, read_slot равен write_slot минус количество тех единиц в write_bitmap, которые "новее" нужной версии;
- если write_bitmap[version] равен 1, чтение производится из слота read_slot;
- если write_bitmap[version] равен 0, чтение производится из слота read_slot минус 1.

В приведенном на рис.4 примере чтение версий 3 и 4 производится из слота 1, а чтение версий 1 и 2 из слота 0.

Переполнение количества версий

Приведенные выше примеры предполагают, что номер текущей версии меньше, чем num_versions. Для того, чтобы можно было работать с бОльшими номерами версий, используются вычисления по модулю num_versions. Битовая карта версий write_bitmap в этом случае представляется как кольцевой буфер (рис.5).

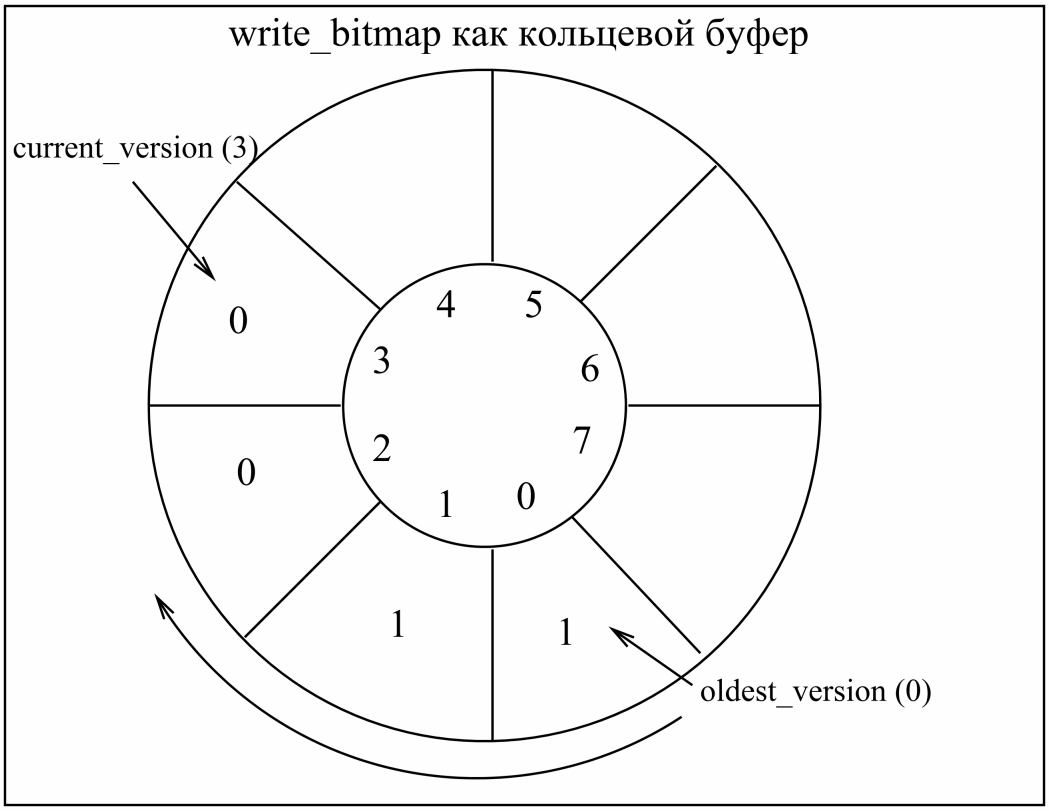

Рис. 5

Головой буфера является разряд current_version по модулю num_versions, а хвостом - oldest_version по модулю num_versions

По мере заполнения кольцевого буфера начинается вытеснение старых версий (рис. 6).

Правила вытеснения старых версий:

если (current_version + 1) mod num_versions == oldest_version mod num_versions

то oldest_version++

current_version++

write_bitmap[current_version mod num_versions]=0

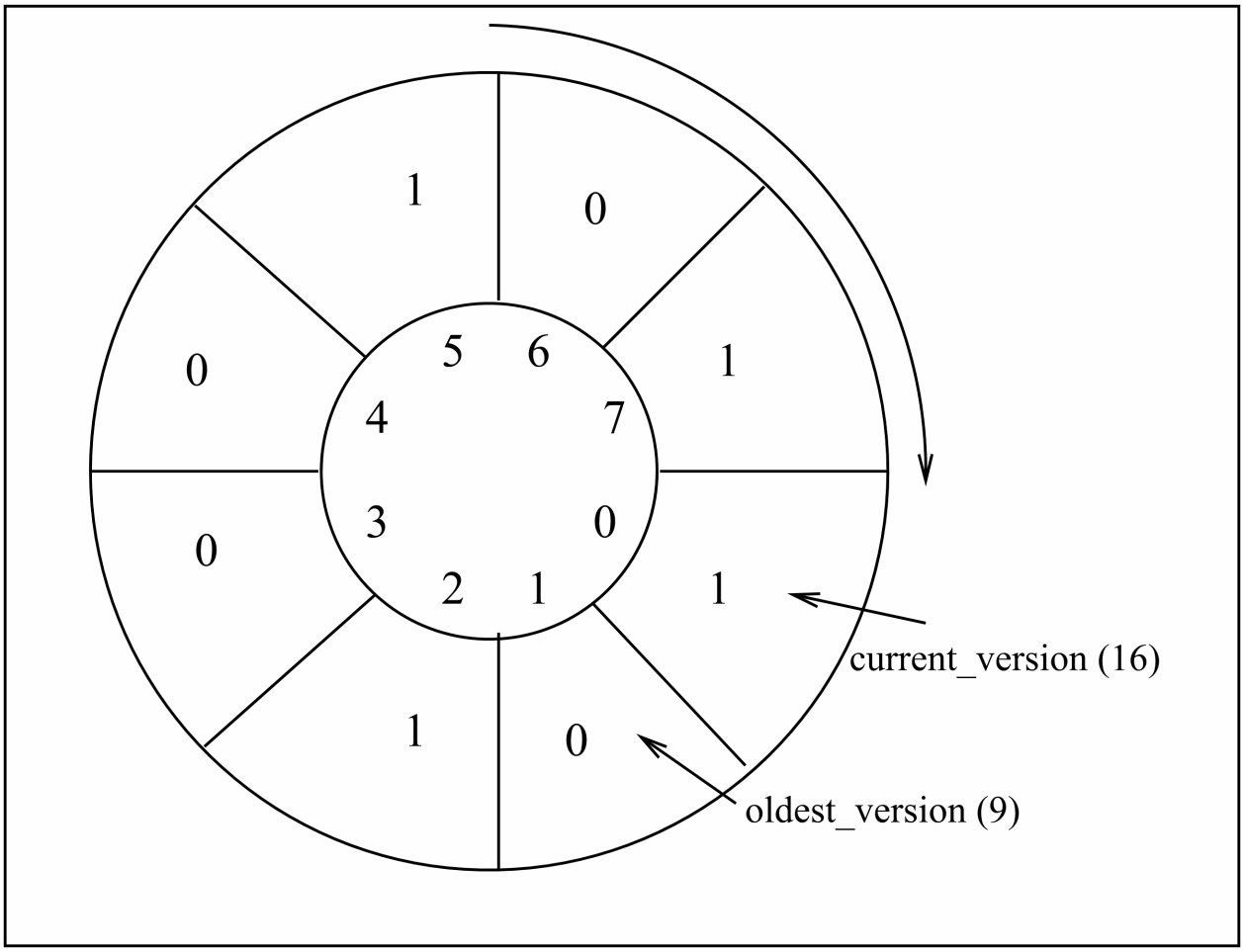

Рис. 6

Оценка потребности в оперативной памяти

Для работы dm_version необходимо хранить в оперативной памяти массив vchunk, размер которого определяется количеством экстентов в LV (NUMLE) и размером структуры vchunk.

Если ограничить число хранимых версий восемью, то размер vchunk можно ограничить двумя байтами:

```
struct vchunk {
   uint8_t write_slot;
   uint8_t write_bitmap;
};
```

Двадцать четыре версии уложатся в четыре байта:

```
struct vchunk {
   uint8_t write_slot;
   uint8_t write_bitmap[3];
};
```

Количество экстентов определяется тремя величинами:

- объемом VG в байтах(VGSIZE);

- количеством версий (NUMVER);
- размером экстента в байтах (LESIZE).

NUMLE= VGSIZE/(LESIZE*NUMVER)

Предположим, используется VG объемом 1ТБ для хранения двух версий при размере экстента 10МБ. В результате, общий размер массива равен 2Б*1000000МБ/(10МБ*2) т.е. 100 КБ, что вполне приемлемо. Для двадцати четырех версий, размер массива равен 4Б*1000000МБ/(10МБ*24) т.е. 16 КБ

При монтировании сохраненных версий LV необходимо создавать копию массива vchunk. Это приводит к увеличению потребности в памяти пропорционально количеству смонтированных версий.

Операции с LV

Создание LV:

- формируется набор параметров: VG для размещения данных, количество версий, размер экстента;
- создается массив vchunk подходящего размера;
- устанавливаются current_version=0; oldest_version=0;
- для всех элементов массива vchunk устанавливаются write_bitmap[0]=1; write_slot=0;
- массив vchunk сохранятся в служебную область на VG.

Создание новой версии LV:

1. устанавливается флаг запрета записи на LV;
2. массив vchunk сохраняется в служебную область на VG;
3. current_version увеличивается на единицу; вычисляется oldest_version;
4. для всех элементов массива vchunk пересчитывается write_bitmap и write_slot;
5. снимается флаг записи на LV.

Откат к произвольной сохраненной версии LV (возможен только в режиме offline):

1. current_version уменьшается на нужную величину;
2. для всех элементов массива vchunk из write_slot вычитается количество тех единиц в write_bitmap, которые "новее" чем восстанавливаемая версия.
3. массив vchunk сохраняется в служебную область на VG

Схема именования LV

Драйвер dm_version регистрирует num_versions интерфейсов блочных устройств. Один интерфейс используется для доступа к текущей версии, остальные - к сохраненным. В момент вытеснения версии закрепленный за ней интерфейс переводится в состояние недоступности, а затем выделяется для обслуживания свежесохраненной версии. За драйверами могут быть закреплены динамически создаваемые имена. Например, если текущая версия LV именуется LogVol, то доступ к сохраненным версиям может производиться по именам LogVol.ver110, LogVol.ver109, LogVol.ver108:

Устойчивость к сбоям

При отключении питания в момент записи данных теряется текущая, не сохраненная версия LV - это предсказуемый результат. После перезагрузки системы в качестве текущей версии загрузится последняя сохраненная.

При отключении питания в момент записи служебной информации возможно ее разрушение. Учитывая размер служебной информации в 100КБ, это маловероятно, но возможно. Для повышения надежности предлагается хранить в служебной области две копии vchunk: для последней и предпоследней версий. В этом случае в качестве текущей версии загрузится последняя корректно сохраненная.

Совместимость с файловыми системами

Драйвер ФС не имеет никакой информации о создании новой версии LV, поэтому при создании новой версии в режиме online целостность файловой системы в сохраняемой версии LV не гарантируется. Для ФС без журналирования это означает необходимость размонтирования ФС для сохранения версии. Для ФС с журналированием возможно создание версий на лету. Поскольку разрешена запись в экстенты измененные в сохраненной версии, то существует и возможность отката операции записи. Следовательно, целостность ФС будет автоматически восстановлена при монтировании.

Другим решением проблемы является принудительная синхронизация ФС перед началом сохранения версии.

Существующие альтернативы

Единственной доступной в Linux ФС с поддержкой версий является Zetabyte File System фирмы SUN [3]. Это великолепная ФС, обладающая большими возможностями, в том числе и возможностью сохранения версий отдельных файлов. К ее недостаткам можно отнести только два:

1) из-за лицензионных проблем она не включена в ядро Linux (но может быть использована через драйвер fuse);
2) невозможна конвертация стандартной ext3fs в ZFS.

Драйвер dm_snapshot позволяет сохранять "снимки" состояния LV, но за счет алгоритма "copy on write" он, как минимум, вдвое уменьшает скорость записи в текущую версию, а при увеличении количества последовательных "снимков" скорость записи падает пропорционально их количеству.

Модификация ext2fs до ФС с сохранением версий файлов, предложенная в [4], не была реализована.

Заключение

На момент написания статьи все описанные алгоритмы и структуры данных реализованы в виде модуля ядра Linux dm_version. После отладки данного модуля необходимо будет:

- протестировать совместимость системы сохранения версий с различными ФС;
- разработать утилиты управления модулем;
- разработать методику конвертации существующих LV в LV с поддержкой версий;
- интегрировать модуль dm_version с системой сохранения контрольных точек программы blcr.

ЛИТЕРАТУРА:

1. http://ru.wikipedia.org/wiki/LVM
2. http://en.wikipedia.org/wiki/Device_mapper
3. http://ru.wikipedia.org/wiki/ZFS
4. А.С.Игумнов, М.А.Стафеев Файловая система с поддержкой контрольных точек// Алгоритмы и программные средства параллельных вычислений. Вып. 8 - Екатеринбург: Уро РАН, 2004. N 8. С.150-162.