Компания **Netwell** - российский дистрибьютор высокотехнологичного оборудования. Основные направления деятельности – сетевые технологии, системы хранения данных, сетевая и информационная безопасность. **Netwell** является **официальным дистрибьютором компании NetApp.**

NETAPP WHITE PAPER

Разработка файловой системы для специализированного файлсервера NFS

Dave Hitz, James Lau, & Michael Malcolm | Network Appliance | WP 3002

Несколько слов сегодняшнему читателю 3

Обзор 3

1. Введение 3

2. Введение в Snapshots 4

2.1. Пользовательский доступ к Snapshots 5

2.2. Управление Snapshot 5

3. Устройство WAFL 6

3.1. Обзор 6

3.2. Метаданные расположены в файлах 6

3.3. Дерево блоков 7

3.4. Snapshots 7

3.5. Целостность файловой системы и Non-Volatile RAM 9

3.6. Write Allocation 11

4. Структуры данных и алгоритмы Snapshot 12

4.1. Block-Map File 12

4.2. Создание Snapshot 12

5. Производительность 13

6. Выводы 14

Библиография 15

Несколько слов сегодняшнему читателю

Network Appliance продолжает разработку своей технологии и продуктов с ее использованием, выпуская новые и улучшенные системы каждые девять месяцев на протяжении, по крайней мере последних трех лет. Этот документ соответствует продуктовой линейке и характеристикам систем на март 1995 года. Базовая архитектура, структура и принципы работы файловой системы WAFL, а также детали реализации RAID-4, описанные здесь, остаются актуальными и по-прежнему используются в системах хранения NetApp, которые в момент публикации этого документа назывались FAServer. Более новый обзор архитектуры систем NetApp можно найти в документе **TR-3014: Multiprotocol Data Access: NFS, CIFS, and HTTP**.

Обзор

Network Appliance недавно начала поставлять новый тип сетевого сервера, под названием NFS file server appliance, который является выделенным сервером, предоставляющим одну функцию – файловый сервис по протоколу NFS. Требования к файловой системе такого серверного устройства несколько отличаются от таковых для UNIX-сервера общего применения, так как сервер NFS должен быть оптимизирован для доступа к данным по сети, и так как мы бы хотели, чтобы такое устройство было бы простым в использовании.

Этот документ описывает WAFL™ (Write Anywhere File Layout – Файловая Структура с Записью Повсюду), которая использована в нем в качестве файловой системы, разработанная специально для серверного NFS-устройства. Основной целью алгоритмов и структур данных, применяемых в WAFL, было использование Snapshots, мгновенных копий данных, являющихся read-only клонами действующей файловой системы. WAFL использует технологию, подобную copy-on-write, чтобы минимизировать используемое снэпшотами пространство дисков. Документ также описывает то, как WAFL использует Snapshots для устранения необходимости проверки целостности файловой системы, в случае нештатного завершения работы.

1. Введение

Используемый в названии компании Network Appliance термин «appliance» обозначает устройство, разработанное для выполнения определенной функции. Важная тенденция в области, например, сетевых систем, заключается в использовании для типовых функций, специально разработанных устройств, вместо использования для этого универсальных компьютеров «общего применения». Простейший пример – роутеры производства Cisco или Bay Networks, пришедшие на смену обычным UNIX-компьютерам с программами, осуществлявшими роутинг IP-пакетов. Другой пример – факсовые сервера или сетевые принтеры.

Новый тип сетевого устройства, «network appliance», это устройство-сервер NFS. Требования к файловой системе, используемой в качестве NFS-устройства, отличаются от файловой системы общего применения: характер и «паттерн» доступа по NFS отличен от такового при локальном доступе, и специализированный характер устройства также влияет на решение, выбранное при создании файловой системы.

WAFL (Write Anywhere File Layout) это файловая система, использованная во всех устройствах сетевого хранения, производства Network Appliance. WAFL была разработана для удовлетворения следующим четырем требованиям:

1. Она должна обеспечивать быстрый сервис NFS.
2. Она должна поддерживать большие файловые системы, динамически растущие по мере добавления дисков в систему.
3. Она должна обеспечивать высокую производительность при использовании RAID.
4. Она должна уметь быстро рестартовать, в том числе после нештатного завершения работы, например в случае отказа по питанию или программной ошибки.

Требование к быстрому сервису NFS очевидно, так как мы используем WAFL в NFS-сервере. Поддержка больших файловых систем упрощает системное администрирование, позволяя всему пространству дисков находиться в одной большой партиции. Большая файловая система, в свою очередь, делает необходимым использование RAID, так как увеличение количества дисков повышает вероятность дисковой ошибки на такой файловой системе. Большая файловая система обычно также требует специальных технологий для осуществления быстрого рестарта, так как функции «file system consistency check» в обычных файловых системах UNIX становятся неприемлемо медленными при увеличении их размеров.

И NFS, и RAID, требовательны к производительности при записи: NFS, так как сервера должны безопасно успевать сохранять данные, до того, как смогут ответить клиенту на его NFS request, а RAID, так как последовательность «чтение-изменение-запись» применяется при использовании контроля четности в типовых RAID-схемах. Эти требования привели нас к необходимости использования так называемой NVRAM (non-volatile RAM), чтобы снизить время отклика для NFS, и использовать метод «write-anywhere», который позволяет WAFL, в свою очередь, писать на диск так, чтобы минимизировать при этом затраты на поддержку RAID. Метод «write-anywhere» позволяет использовать снэпшоты (Snapshots), которые, в том числе, устраняют требование проводить длительную проверку целостности файловой системы после потери питания или системной ошибки.

2. Введение в Snapshots

Одним из основных и принципиальных свойств WAFL являются так называемые «Snapshots», которые являются read-only копиями всей файловой системы. WAFL создает и удаляет Snapshots автоматически, в назначенное расписанием время, и хранит до 255 Snapshots каждого тома, обеспечивая простой доступ к старым версиям файлов.

Snapshots используют технологию, подобную copy-on-write, чтобы избежать дублирования блоков данных на дисках, используемых в Snapshot, также как и активной файловой системе. Только когда блок в активной файловой системе изменен или удален, тогда Snapshots, содержащий эти блоки, начинает занимать место на диске.

Пользователь может получать доступ к Snapshots по NFS, чтобы восстанавливать файлы, которые были случайно удалены или изменены, а системные администраторы могут использовать Snapshots для создания резервной копии работающей системы. Кроме этого, WAFL и внутри себя

использует Snapshots, чтобы обеспечить быстрый рестарт, в случае нештатного выключения или иного случая некорректного завершения работы.

2.1. Пользовательский доступ к Snapshots

Каждая директория в файловой системе содержит скрытую поддиректорию по имени `.snapshot`, которая позволяет пользователям получать доступ к содержимому снэпшотов через NFS. Допустим, что пользователь случайно удалил файл по имени `todo` и хочет вернуть его. Следующий пример покажет, как посмотреть все сохраненные версии `todo` находящиеся в Snapshots:

```
spike% ls -lut .snapshot/*/todo
-rw-r--r-- 1 hitz 52880 Oct 15 00:00
.snapshot/nightly.0/todo
-rw-r--r-- 1 hitz 52880 Oct 14 19:00
.snapshot/hourly.0/todo
-rw-r--r-- 1 hitz 52829 Oct 14 15:00
.snapshot/hourly.1/todo
...
-rw-r--r-- 1 hitz 55059 Oct 10 00:00
.snapshot/nightly.4/todo
-rw-r--r-- 1 hitz 55059 Oct 9 00:00
.snapshot/nightly.5/todo
```

С опцией `-u`, команда `ls` покажет для файла `todo` значение времени доступа, которое будет являться временем создания его снэпшота. Пользователь может восстановить нужную ему версию файла `todo`, скопировав его из поддиректории снэпшотов назад, в свою текущую директорию:

```
spike% cp .snapshot/hourly.0/todo .
```

Директория `.snapshot` сделана скрытой ("hidden"), для того, чтобы она не показывалась при обычном выводе содержимого пользовательской директории. Если `.snapshot` будет видимой, то команды типа `fin` будут показывать значительно больше файлов, чем ожидает увидеть их пользователь в директории, а команда `rm -rf` будет завершаться с ошибкой, так как файлы в Snapshots являются read-only и не могут быть удалены обычным способом.

2.2. Управление Snapshot

FAServer использует специальные команды, позволяющие системным администраторам создавать и удалять Snapshots, но оно также может создавать и удалять большинство снэпшотов автоматически. По умолчанию, FAServer создает четыре «часовых» снэпшота, в различные моменты времени рабочего дня, а также «ночной» снэпшот в полночь. Он хранит такие «часовые» снэпшоты в течение двух дней, а «ночные» в течении недели. Он также создает «недельный» снэпшот в полночь воскресенья, хранящийся две недели.

Для файловой системы, содержимое которой изменяется быстро, это расписание может начать использовать слишком много пространства на диске, и может потребоваться удалять снэпшоты чаще. Снэпшоты полезны даже если они хранятся всего несколько часов, так как пользователи чаще всего спохватываются об удаленном нужном файле довольно скоро. Для файловой системы, которая изменяется медленно, будет разумным хранить снэпшоты подольше. В типичной системе, хранение Snapshots в течении недели использует от 10 до 20 процентов пространства дисков.

3. Устройство WAFL

3.1. Обзор

WAFL это UNIX-совместимая файловая система, оптимизированная для сетевого файлового доступа. Во многом WAFL похожа на другие файловые системы из мира UNIX, такие как Berkeley Fast File System (FFS) и TransArc's *Episode* file system. WAFL это блочная файловая система, использующая inodes для организации и описания файлов. Она использует блоки, размером 4 KB, без их фрагментирования.

Каждый inode WAFL содержит 16 блоков-указателей, для того, чтобы показать, какие блоки принадлежат соответствующему файлу. В отличие от FFS, все блоки-указатели в WAFL inode ссылаются на блоки на том же уровне. Таким образом, inodes для файлов меньше, чем 64 KB, используют 16 блоков-указателей, чтобы указать на блоки данных. Inodes для файлов меньше, чем 64 MB, указывают на промежуточные (indirect) блоки, которые уже указывают на блоки собственно данных. Inodes для больших файлов указывают на удвоенные промежуточные блоки. Для очень маленьких файлов, данные хранятся непосредственно в самом inode, в месте размещения блоков-указателей.

3.2. Метаданные расположены в файлах

Как и в *Episode*, WAFL хранит метаданные в файлах. Три файла метаданных WAFL это файл inode, содержащий inodes файловой системы, файл block-map, который перечисляет свободные блоки, и файл inode-map, указывающий свободные inodes. Мы пользуемся термином «block map» вместо привычного «bit map», так как эти файлы используют более чем один бит на каждый элемент. Формат файла block-map описан подробнее ниже.

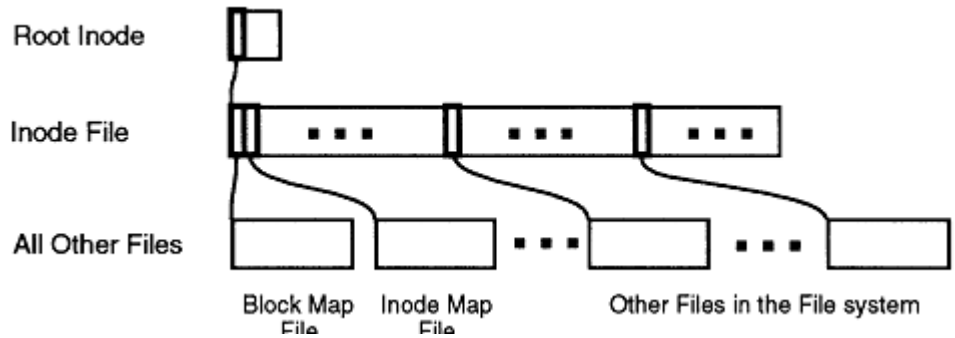

Рис 1: Файловая система WAFL как дерево блоков, с корневым inode, который описывает inode file, на самом верху, файлов meta-data и обычных файлов, лежащих ниже.

Хранение метаданных в файлах позволяет WAFL писать блоки метаданных повсюду на диске. Это и есть источник имени WAFL, акронима от Write Anywhere File Layout – Файловая Структура с Повсеместной Записью. Схема с write-anywhere позволяет WAFL эффективно работать с RAID, организуя множественные записи в один страйп RAID, и устраняя проблему низкой производительности при записи «классического» RAID с четностью, когда одна операция записи превращается в несколько из-за необходимости изменить только один блок этого страйпа.

Хранение метаданных в файлах позволяет легко увеличивать размер файловой системы «на лету». Когда добавляется новый диск, FAServer автоматически увеличивает размер файлов метаданных. Системный администратор может увеличить число inodes в файловой системе вручную, если их

количество по умолчанию недостаточно. Ну и наконец, режим write-anywhere позволяет не использовать технологию copy-on-write для создания снэпшотов. Для того чтобы Snapshots работали правильно, WAFL должен уметь писать все новые данные, включая метаданные, в новое место на диске, вместо перезаписи старых данных. Если бы WAFL хранила метаданные в фиксированных местах диска, это было бы невозможно.

3.3. Дерево блоков

Файловую систему WAFL правильнее всего представить себе как дерево блоков. В корне такого дерева находится так называемый root inode, как это показано на Рис 1. Root inode это специальный inode который описывает inode file. В свою очередь inode file содержит inodes, которые описывают все остальные файлы файловой системы, включая файлы block-map и inode-map. «Листья» этого «дерева» это блоки данных всех файлов.

Рис 2 это более детальная версия Рис 1. Он показывает, что файлы состоят из индивидуальных блоков, и большие файлы используют дополнительный слой промежуточных блоков-указателей между inode и блоками данных такого файла. При загрузке системы нам важно найти этот «корень», поэтому единственное исключение из главного правила WAFL «писать повсюду», это вот этот root inode. Он пишется на фиксированное место на диске, чтобы WAFL могла найти его при старте.

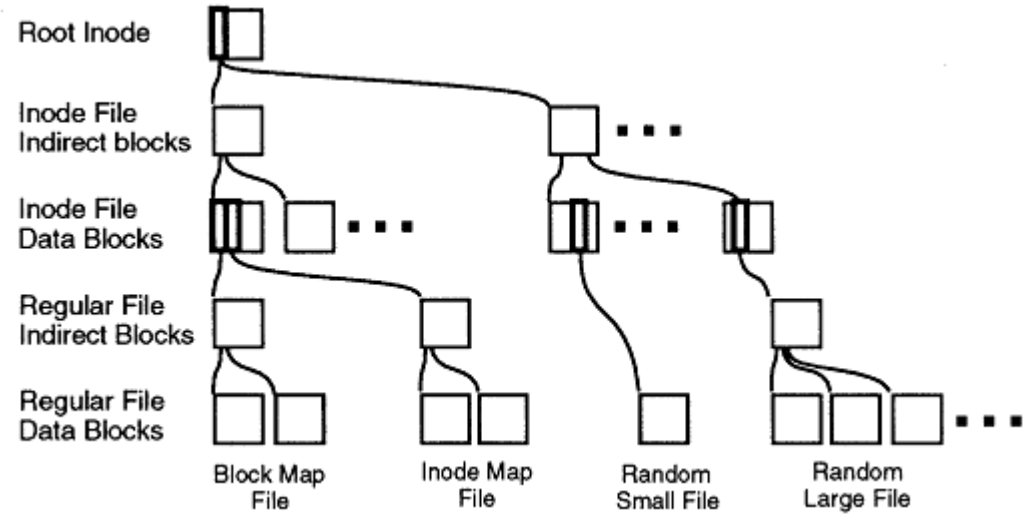

Рис 2: Более детальный рисунок структуры дерева блоков WAFL.

3.4. Snapshots

Понимание того, что файловая система WAFL это дерево блоков, с «корнем» в root inode есть ключ к пониманию Snapshots. Для создания виртуальной копии этого дерева блоков, WAFL просто дублирует root inode. Рис 3 показывает, как это работает.

Рис 3(a) это упрощенная диаграмма файловой системы с Рис 2, где опущены внутренние элементы дерева, такие как inodes и промежуточные блоки.

Рис 3(b) показывает, как WAFL создает новый Snapshot, делая копию root inode. Эта копия inode становится корнем дерева блоков, соответствующих Snapshot, также как root inode является корнем для active file system. Когда создан Snapshot inode, он указывает в точности на те же самые блоки диска, что и root inode, так что только что созданный Snapshot не занимает пространства на диске, кроме собственно блока со Snapshot inode.

Рис 3(c) показывает, что происходит, когда пользователь изменяет данные в блоке D. WAFL пишет новые данные в блок D' на диске, и изменяет active file system, чтобы она указывала на новый блок. При этом Snapshot по-прежнему ссылается на исходный блок D, оставшийся неизмененным на диске. Со временем, по мере того, как файлы на активной файловой системе изменяются или удаляются, Snapshot ссылается на все больше и больше блоков, которые более не используются в активной файловой системе. Степень, с которой изменяются файлы, определяет то, как долго могут храниться снэпшоты, прежде чем они начнут занимать на диске неприемлемо много места.

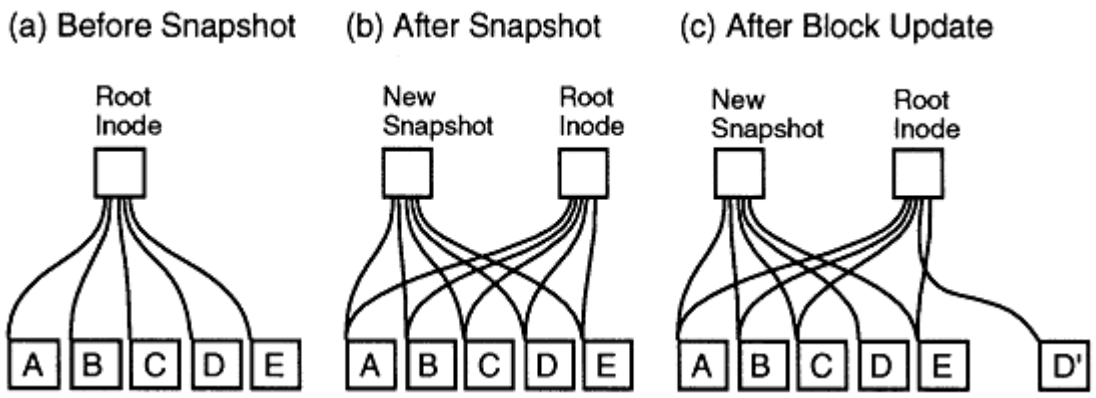

Рис 3: WAFL создает Snapshot дублируя root inode. WAFL не изменяет блоки в Snapshot, записывая новые и измененные данные в новое место диска.

Любопытно сравнить снэпшоты WAFL и fileset clones файловой системы *Episode*. Вместо дублирования root inode, *Episode* создает клон, копируя весь inode file. Это создает заметный трафик ввода-вывода, и использует много места на диске. Например, файловая система размером 10 GB с одним inode на каждые 4 KB дискового пространства займет 320 MB только одних inodes. В такой файловой системе, создание Snapshot копированием inodes создаст трафик ввода-вывода в 320 MB и займет 320 MB пространства на диске для каждого созданного снэпшота. Создание 10 таких Snapshots займет почти треть пространства файловой системы, даже еще до того, как начнут изменяться блоки данных.

Копируя только один root inode, WAFL создает Snapshots очень быстро, и с небольшим трафиком ввода-вывода. Производительность при создании Snapshot очень важна для WAFL, так как она создает Snapshot каждые несколько секунд, для того, чтобы обеспечить быстрое восстановление при возможном некорректном отключении системы.

Рисунок 4 показывает переход из Рис 3(b) к 3(c) более детально. Когда дисковый блок должен измениться, и его содержимое пишется на новое место, то «родитель» этого блока должен быть также изменен, чтобы отразить это новое расположение. «Родитель родителя», в свою очередь, должен быть записан на новое место, и так далее, до самого корня дерева.

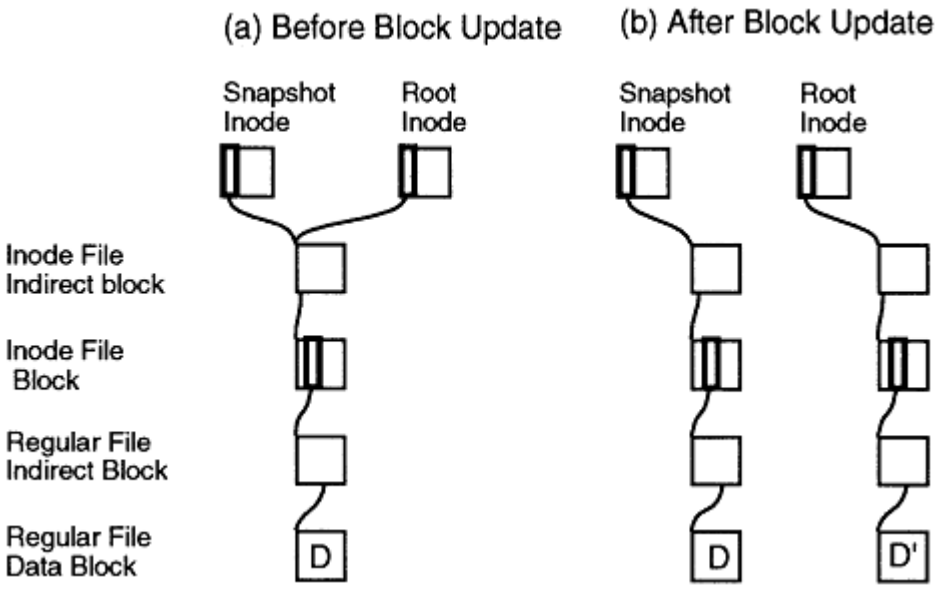

Рис 4: При записи блока на новое место, указатели на него в вышележащих блоках должны быть изменены, что также вызывает их перезапись в новое место.

WAFL была бы очень неэффективной, если бы перезаписывала все эти блоки при каждом запросе NFS на запись. Вместо этого, WAFL собирает вместе несколько сотен запросов NFS перед тем, как инициирует сеанс записи. Во время этого сеанса записи, WAFL выделяет пространство диска для всех измененных данных в кэше («dirty data») и назначает время для проведения операции записи. В результате, связанные изменения блоков, такие как изменения в промежуточных (indirect) блоках, и блоках в inode file, пишутся один раз на такой сеанс записи, вместо одного раза на каждый NFS request.

3.5. Целостность файловой системы и Non-Volatile RAM

WAFL устраняет необходимость проводить проверку целостности файловой системы в случае некорректного выключения, создавая специальный внутренний Snapshot под названием «consistency point» каждые несколько секунд. В отличие от прочих снэпшотов, consistency point не имеет имени, и не доступен через NFS. Как и все прочие снэпшоты, consistency point это полный, непротиворечивый образ всей файловой системы. Когда WAFL перезапускается, она просто откатывается на самый новый по времени создания consistency point. Это позволяет FAServer целиком перезагружаться не более чем за минуту, даже, например, при 20 GB данных в одной партиции[*].

Между моментами создания consistency points, WAFL пишет данные на диск, но пишет только в блоки, которые не используются, так что дерево блоков, соответствующее самой свежей consistency point остается неизменным. WAFL обрабатывает сотни и тысячи запросов NFS между consistency points так, что находящийся на диске образ файловой системы остается неизменным на протяжении тех нескольких секунд, пока WAFL не запишет новый consistency point, и в этот момент, образ файловой системы на диске атомарно переходит в новое состояние, отражающее изменения, сделанные новым запросом. Хотя такая технология и необычна для файловых систем

[*] Оригинал статьи написан в 1994 году

мира UNIX, она хорошо известна в мире баз данных. Но даже для баз данных необычно писать такое множество операций в один присест, как это делает WAFL при создании consistency points.

WAFL использует специальное аппаратное устройство, non-volatile RAM (NVRAM) для хранения логов операций NFS, проведенных с момента последней consistency point. (NVRAM это специальная память с независимым батарейным питанием, хранящая данные даже если основное питание системы отсутствует). В случае некорректного выключения, WAFL перевоспроизводит все запросы в логе, чтобы сохранить их от потери. Когда FAServer завершает работу нормально он создает последнюю consistency point после остановки сервиса NFS. Следовательно, в случае корректного завершения, NVRAM не содержит никаких необработанных операций NFS, и выключена, чтобы продлить срок жизни батареи.

WAFL делит NVRAM на два отдельных лога. Когда один лог заполняется, WAFL переключается на второй, и начинает запись consistency point, чтобы сохранить изменения первого лога на диске. WAFL назначает создание consistency point каждые 10 секунд, даже если лог и не заполнен, чтобы держать файловую систему на дисках не слишком отставшей от текущего состояния.

Запись операций NFS в NVRAM имеет несколько преимуществ перед традиционными методами использования NVRAM для кэширования записей на уровне дисков. Lyon и Sandberg описывали технологию использования NVRAM write cache, использованной в Legato Prestoserve NFS accelerator.

Обработка операций NFS и кэширование получившихся дисковых записей занимает, в общем случае, гораздо больше пространства в NVRAM, чем простое логирование информации, необходимой для воспроизведения операции. Например, для перемещения файла из одной директории в другую, файловая система должна обновить содержимое inodes как директории-источника, так и получателя. В FFS, в которой блоки размером по 8 KB, это займет 32 KB пространства кэша. WAFL использует только около 150 байт, чтобы записать информацию, необходимую для воспроизведения операции переименования. Более чем 200-кратная разница в объемах это, конечно, предельный случай, но даже в случае простой записи 8 KB, кэширование блоков диска займет 8 KB для данных, 8 KB на обновление inode, и для больших файлов, по меньшей мере, еще 8 KB для indirect block. WAFL записывает всего 8 KB данных, и примерно 120 байт заголовочной информации. В типичном наборе операций NFS, WAFL может хранить более 1000 операций на мегабайт NVRAM.

Использование NVRAM как традиционного кэша незаписанных дисковых блоков (write-back cache) превращает ее в важную и неотъемлемую часть дисковой подсистемы. Ошибка в NVRAM может повредить файловую систему так, что `fsck` не сможет обнаружить или восстановить это. Однако если что-то пошло не так с NVRAM, как ее использует WAFL, то WAFL может потерять несколько операций NFS, но лежащий на дисках образ файловой системы остается полностью непротиворечивым. Это важный момент, почему NVRAM надежен, но не обязан быть столь же надежен, как RAID-массив.

Ну и наконец, важное преимущество логирования операций NFS, это улучшение времени отклика NFS. Для ответа на запрос NFS, файловая система без NVRAM должна обновить структуры в памяти, выделить место на диске под новые данные и подождать, пока все данные достигнут дисков. Файловая система с кэшированием данных в NVRAM делает все эти шаги, кроме того, что вместо ожидания записи данных на диск она записывает их в NVRAM. WAFL может воспроизвести операцию NFS гораздо быстрее, так как ей нужно только обновить данные в памяти и записать

операцию в лог. Не нужно выделять место на диске под новые данные, или копировать измененные данные в NVRAM.

3.6. Write Allocation

Производительность на запись особенно важна для сетевого файлового сервера. Ousterhout рассматривал ситуацию, что когда кэши чтения достаточно велики на обоих концах, как на клиенте, так и на сервере, запись начинает превалировать в вводе-выводе системы [Ousterhout89]. Этот эффект особенно сказывается в случае NFS, которая имеет очень маленький кэш на запись на клиентской стороне. В результате, диски NFS-сервера получают почти впятеро больше операций записи, чем чтения.

Разработка WAFL в значительной мере основывалась на желании увеличить гибкость политик write allocation policies. Эта гибкость представляется в трех видах:

(1) WAFL может писать любой блок файловой системы (кроме одного, содержащего root inode) в любое место на диске. В FFS, meta-data, такие, как inodes и bit maps, хранятся в фиксированных местах на диске. Это мешает оптимизировать записи в FFS, например, поместить данные обновленного файла и его inode рядом на диске. Так как WAFL может писать meta-data куда угодно на диске, она может оптимизировать запись более творчески.

(2) WAFL может писать блоки на диск в любом порядке. FFS пишет блоки на диск в определенном порядке, так, чтобы fsck(8) мог восстановить целостность файловой системы в случае некорректного выключения. WAFL может писать блоки в любом порядке, так как находящийся на диске образ файловой системы изменяется только в момент, когда WAFL записывает consistency point. Одно ограничение только в том, что WAFL должен записать все блоки в новой consistency point до того, как будет записан root inode для этой consistency point.

(3) WAFL может выделять дисковое пространство для множества операций NFS за один прием, в одном сеансе записи. FFS выделяет место на дисках в процессе каждого NFS-запроса. WAFL собирает сотни запросов NFS перед тем, как записать consistency point, и в это время она может выделить блоки для всех запросов записи в момент создания consistency point. Отложенное выделение пространства под запись улучшает показатели латентности операций NFS удаляя процесс выделения дискового пространства из пути обработки запроса, а также устраняя затраты времени на выделение места под блок, который оказывается удаленным следующей операцией, прежде чем он попадет на диск. Эти особенности дают WAFL исключительную гибкость во write allocation policies. Возможность планировать записи для множества операций за один прием позволяют разрабатывать гораздо более интеллектуальные allocation policies, и факт того, что блоки могут писаться в любое место, в любом порядке, позволяет широкий выбор стратегий. Легко попробовать использовать новую стратегию block allocation без изменений в собственно структурах WAFL на дисках.

Детальное рассмотрение политик записи WAFL выходит за рамки этого документа. Вкратце, WAFL улучшает производительность RAID, записывая множество блоков в единый страйп (stripe); WAFL снижает время позиционирования (seek time) записывая блоки в места, которые находятся один рядом с другим на диске; кроме того WAFL снижает перепозиционирование головок при чтении больших файлов, помещая последовательные блоки в файл на одном диске в RAID-массиве. Оптимизация write allocation непроста, так как эти цели часто конфликтуют.

4. Структуры данных и алгоритмы Snapshot

4.1. Block-Map File

Большинство файловых систем отслеживают свободные блоки, используя так называемую «битовую карту», bit map, в которой каждому дисковому блоку соответствует бит в файле. Если бит установлен – блок используется. Эта технология не работает в WAFL, так как несколько снэпшотов могут ссылаться на один и тот же блок одновременно. Файл block-map в WAFL содержит 32-bit записи для каждого дискового блока в 4 KB. Бит 0 устанавливается, если активная файловая система ссылается на этот блок, бит 1 устанавливается, если первый Snapshot ссылается на тот же блок, и так далее. Блок считается используемым, если установлен хотя бы один бит в его записи в файле block-map.

Рисунок 5 показывает «жизненный цикл» типичной записи в block-map. В момент времени **t1**, запись в block-map полностью чиста, показывая, что блок свободен. В момент **t2**, WAFL выделяет блок для использования, и сохраняет в нем данные. Когда создается Snapshots, в моменты **t3** и **t4**, WAFL копирует бит активной файловой системы в бит, показывающий участие в Snapshot. Блок удаляется из активной файловой системы в момент **t5**. Это получается, когда, например, файл, содержащий этот блок, удален, или содержимое блока обновлено, и поэтому перезаписано в новое место диска, в новый блок. Однако блок не может быть использован для хранения данных повторно, пока на него ссылаются из Snapshot. В момент **t8**, когда использовать блок прекращает и активная файловая система, и снэпшоты, блок считается свободным.

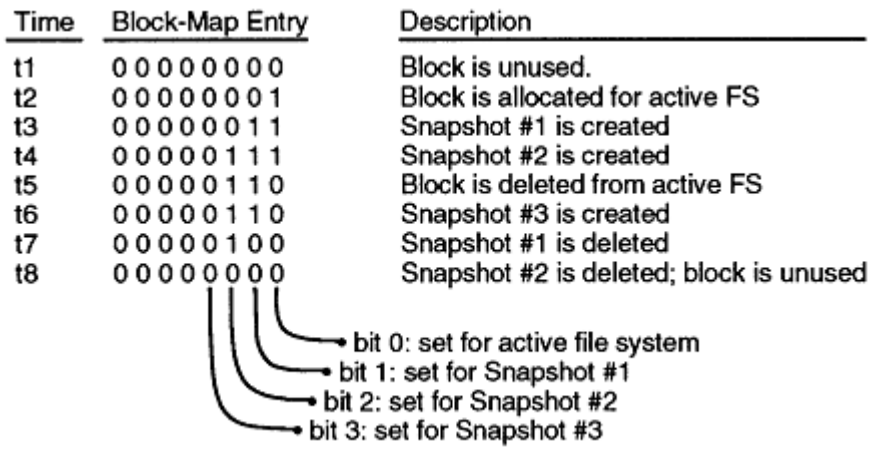

Рис 5: Жизненный цикл записи в block-map file.

4.2. Создание Snapshot

Проблема, которую следует решить при записи Snapshot на диск, это необходимость избежать локирования входящих операций NFS. Проблема заключается в том, что новый запрос NFS может захотеть изменить закэшированные данные, которые являются частью снэпшота, и которые должны остаться неизменными, пока не попадут на диск. Простым решением было бы приостановить NFS на время создания снэпшота, записать снэпшот, и продолжить обработку NFS. Однако запись снэпшота может занять секунду, что очень долго для приостановки ответа NFS-сервера. Помните, что WAFL создает снэпшот в виде consistency point каждые 10 секунд как минимум, так что производительность тут критична.

Метод WAFL, чтобы сохранить данные Snapshot непротиворечивыми (self-consistent), это пометить все измененные данные (dirty data) в кэше, как IN_SNAPSHOT. Правило в момент создания

Snapshot таково, что данные, помеченные IN_SNAPSHOT не должны изменяться, и данные, не помеченные IN_SNAPSHOT не должны сбрасываться на диск. Операции NFS могут читать все данные файловой системы, и они могут изменять данные, которые не отмечены IN_SNAPSHOT, но обработка операций, которым нужно изменить данные IN_SNAPSHOT должна быть задержана.

Чтобы избежать локирования операции NFS, WAFL должен сбросить данные IN_SNAPSHOT на диск так быстро, как это возможно. Чтобы сделать это, WAFL производит следующие шаги:

(1) Выделяется место на диске для всех файлов, содержащих блоки, отмеченные IN_SNAPSHOT. WAFL кэширует данные inode в двух местах: в специальном кэше in-core inodes, и дисковых буферах, принадлежащих файлу inode. Когда осуществлено выделение места под запись файла, WAFL копирует обновленную информацию о inode из inode-кэша в соответствующий дисковый буфер (inode file disk buffer), и очищает бит IN_SNAPSHOT в in-core inode. Когда выполнен этот шаг, то никакие из inodes обычных файлов не отмечены как IN_SNAPSHOT, и большинство операций NFS могут продолжать выполняться без блокирования. К счастью, этот шаг может быть выполнен очень быстро, так как не требует выполнения дискового ввода-вывода.

(2) Обновляется block-map file. Для каждой записи в block-map, WAFL копирует бит для активной файловой системы в бит для нового снэпшота.

(3) Записываются все дисковые буфера IN_SNAPSHOT в кэше, в их новые, выделенные ранее места на диске. Когда буфера сброшены, WAFL рестартует все те операции NFS, что ожидали возможности их модифицировать.

(4) Дублируется root inode для создания inode, соответствующего новому Snapshot, и выключается бит IN_SNAPSHOT для root inode. Inode для нового Snapshot не должен быть записан на диск, пока не записаны все другие блоки в Snapshot. Если не следовать этому правилу, то неожиданное выключение может оставить снэпшот в неконсистентном состоянии.

Когда inode для нового Snapshot записана, данных с IN_SNAPSHOT больше нет в кэше, и любые операции NFS, которые все еще задержаны, могут быть продолжены. При нормальной загрузке, WAFL производит эти четыре шага, менее чем за секунду. Шаг (1) может быть выполнен, в общем случае, на несколько сотых секунды, и когда WAFL выполнит его, очень немногие из операций NFS потребуют задержки.

Удаление снэпшота тривиально. WAFL просто обнуляет корневой inode, соответствующий снэпшоту, и очищает бит, соответствующий снэпшоту, в каждой записи block-map.

5. Производительность

Трудно сравнивать производительность WAFL с другими системами напрямую. Так как WAFL работает только в NFS-сервере, то можно сравнивать ее с другими системами только в контексте NFS. Лучший бенчмарк NFS, доступный сегодня, это SPEC SFS (System File Server), известный ранее как LADDIS. Имя LADDIS получено из имен компаний, разрабатывавших этот бенчмарк: Legato, Auspex, Digital, Data General, Interphase, и Sun.

Тест SPEC SFS оценивает производительность NFS, измеряя время отклика сервера при различных величинах нагрузки. Сервера обычно обрабатывают запросы быстрее при невысоком уровне

загрузки; когда загрузка растет, то растет и их время отклика. Результаты производительности для многих популярных NFS-систем хранения можно найти на вебсайте http://www.spec.org

Использование бенчмарка системного уровня, такого как SPEC SFS, для оценки производительности файловой системы может быть не вполне корректно. Кто-то может указать, например, что в приведенных примерах FAServer cluster имеет только 8 файловых систем, в то время, как у других их используется множество. Кроме этого, FAServer всегда использует RAID (который обычно уменьшает производительность файловой системы при небольших операциях, характерных для NFS), в то время как другие системы не используют RAID при измерении своих результатов.

С другой стороны, некоторые могут указать, что FAServer специально разработан под задачи обслуживания NFS, и частично его результаты вызваны специальной оптимизацией под задачи NFS, а не только лишь хорошей производительностью WAFL как файловой системы.

По причине такой весьма специальной природы WAFL, по-видимому, просто не существует честного способа сравнить его производительность с файловыми системами общего применения, но система в целом, вместе с NFS и RAID, может быть вполне удовлетворяющей по своим показателям.

6. Выводы

WAFL был разработан, и вышел в stable, удивительно быстро для новой файловой системы. На момент написания статьи[*] он используется как production file system свыше года, и нам неизвестно ни одного случая потери данных по его вине.

Мы полагаем, что эта стабильность, в значительной мере, является следствием использования метода consistency points. Обработка запросов файловой системы проста, так как WAFL изменяет только структуры данных в памяти и NVRAM log. Метод consistency points устраняет ограничение на порядок дисковых записей, которые являются значительным источником ошибок в большинстве файловых систем. В WAFL код, отвечающий за запись consistency point, сосредоточен в единственном файле, сравнительно мало взаимодействует с остальной WAFL, и вызывается относительно нечасто.

Но мы думаем, что более важным является то, что разработка программного обеспечения высококачественной, высокопроизводительной системы гораздо проще для специализированного устройства, чем для системы общего применения. В сравнении с файловой системой общего применения, WAFL обрабатывает типовые и довольно простые наборы запросов и операций. Обычная компьютерная система принимает запросы от тысяч различных приложений, с различным характером доступа, а новые приложения добавляются достаточно часто. Напротив, WAFL принимает запросы только от клиента NFS. Существует немного реализаций клиента NFS, и новые реализации относительно редки. Конечно, приложения являются источником операций NFS, но код клиента NFS преобразует системные операции в типовые паттерны операций сетевых запросов, и он фильтрует ошибочные операции до того, как они попадут на сервер. Небольшое количество операций, поддерживаемое WAFL делает

[*] Март 1995 года

возможным определить и протестировать весь диапазон возможных вариантов, которые понадобится ей обрабатывать.

Эти преимущества применимы к любому устройству, не только к файловому серверу. Использование специального сетевого устройства (network appliance) имеет смысл только для хорошо известных и широко распространенных протоколов, но для таких протоколов, самостоятельное устройство (appliance) может продемонстрировать значительное превосходство над компьютерной системой общего применения.

Библиография

[Astrahan76] M. Astrahan, M. Blasgen, K. Chamberlain, K. Eswaran, J. Gravy, P. Griffiths, W. King, I. Traiger, B. Wade and V. Watson. System R: Relational Approach to Database Management. ACM Transactions on Database Systems 1, 2 (1976), pp. 97-137.

[Chutani92] Sailesh Chutani, et. al. The Episode File System. Proceedings of the Winter 1992 USENIX Conference, pp. 43-60, San Francisco, CA, January 1992.

[Hitz93] Dave Hitz. An NFS File Server Appliance. Network Appliance, Inc.

[Lyon89] Bob Lyon and Russel Sandberg. Breaking Through the NFS Performance Barrier. SunTech Journal 2(4): 21-27, Autumn 1989.

[McKusick84] Marshall K. McKusick. A Fast File System for UNIX. ACM Transactions on Computer Systems 2(3): 181-97, August 1984.

[Ousterhout89] John Ousterhout and Fred Douglis Beating the I/O Bottleneck: A Case for Log-Structured File Systems. ACM SIGOPS, 23, January 1989.

[Patterson88] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). ACM SIGMOD 88, Chicago, June 1988, pp. 109-116.

[Sandberg85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network File System. Proceedings of the Summer 1985 USENIX Conference, pp. 119- 30, Portland, OR, June 1985.