Динамический анализ и оптимизация ввода-вывода в среде виртуализации GNU Linux/QEMU/KVM

А.Б. Бетелин[1], Г.А. Прилипко[2], А.Г. Прилипко[3], С.Г. Романюк[4], Д.В. Самборский[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ab@niisi.msk.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, prilipko@niisi.msk.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aleksey.prilipko@gmail.com;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, sgrom@niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, samborsky_d@fastmail.com

Аннотация. В данной статье приведены результаты тестирования производительности ввода-вывода виртуальных операционных систем в среде виртуализации GNU Linux/QEMU/KVM и предложены способы увеличения производительности приложений виртуальных ОС с помощью более оптимального использования файлового кэша ОС Linux. Разработана утилита динамического анализа и оптимизации использования виртуальных дисков. Тестирование показало, что использование данной утилиты позволяет достичь более чем двукратного ускорения смешанных нагрузок ввода-вывода на SSD-накопителях.

Ключевые слова: виртуализация, ввод-вывод данных, QEMU, KVM, Linux, vmtouch, vmprobe, mincore

1. Введение

Система виртуализации QEMU/KVM [1, 2] в операционной системе GNU Linux предлагает несколько режимов подключения виртуальных или реальных устройств накопителей данных (так называемых «дисковых» устройств). Накопителем данных для виртуальной ОС может выступать обычный файл, дисковый том, SATA- или SCSI-устройство, NVMe-накопитель данных. Подключение файла или дискового тома в качестве виртуального накопителя данных осуществляется либо в режиме эмуляции некоторого протокола (SATA, SCSI), либо с помощью драйвера семейства VirtIO, использующего технологию паравиртуализации, который должен быть установлен в виртуальную ОС. В последнем случае VirtIO-драйвер напрямую указывает буфер данных и команду записи или чтения библиотеке гипервизора, которая выполняет чтение или запись без лишних накладных расходов.

Для делегирования виртуальной ОС монопольного доступа к накопителю данных используются драйверы vhost-scsi, vhost-user, vfio-pci. Такое подключение накопителей данных обеспечивает максимальную общую производительность и минимальное время выполнения отдельных команд ввода-вывода. Но при этом усложняется настройка сервера виртуализации и становятся невозможными функции моментальных бэкапов (live snapshot), безостановочной миграции (live migration), управление скоростью ввода-вывода (IO throttling) и др.

Поэтому при проектировании вычислительного кластера для среды виртуализации QEMU/KVM приходится находить компромисс между требуемой скоростью ввода-вывода и гибкостью администрирования. Обычно, если не требуется достичь 100% скорости накопителя, то для локальных накопителей оказывается оптимальным использование драйверов семейства VirtIO. Так, в статье [3] было показано, что даже при использовании быстрого накопителя Intel Optane 900P драйвер virtio-scsi с выделенным потоком управления ввода-вывода (iothread) показывает скорость записи блоков данных размером 4Кб всего на 25% меньше, чем при использовании драйвера vhost-user и библиотеки Intel SPDK для прямого доступа к устройству. В случае блоков размером 2Мб разницы в производительности между virtio-scsi и vhost-user не наблюдается.

При подключении виртуальных дисков с помощью VirtIO драйверов система QEMU использует стандартный стек ввода-вывода ядра Linux, поэтому при работе виртуального диска может быть задействован системный файловый кэш (называемый «page cache» в терминологии ядра Linux). Согласно некоторым популярным рекомендациям кэширование SSD-накопителей не ускоряет доступ к данным, и его советуют выключать. Тем не менее, любое подобное общее утверждение рекомендуется проверять тщательным тестированием. Чтобы исследовать влияние

файлового кэша на производительность виртуальных дисков, авторами были выполнены соответствующие тесты и предложены рекомендации по увеличению скорости ввода-вывода виртуальных ОС QEMU/KVM.

Для ускорения ввода-вывода с помощью более оптимального использования файлового кэша была разработана утилита vmdisktouch, выполняющая динамический анализ использования виртуального диска и своевременную загрузку часто используемых областей диска в файловый кэш. На момент публикации авторы не имеют сведений о наличии другого общедоступного инструмента с функциями, аналогичными функциям утилиты vmdisktouch.

2. Конфигурация среды виртуализации QEMU

В данной работе были рассмотрены следующие конфигурации серверов:

- 1 или 2 процессора семейства Intel Xeon;
- от 128Гб до 512Гб оперативной памяти;
- несколько SSD- и HDD-накопителей данных, объединенных в RAID1- или RAID10- массивы.

Эта конфигурация соответствует серверу средней ценовой категории и оптимальна по соотношению цены оборудования к производительности виртуализированных приложений, если нет необходимости в безостановочной миграции виртуальных ОС. На сервере подобной конфигурации могут работать несколько виртуальных ОС, выполняющих функции файл-серверов, серверов приложений и программных сетевых маршрутизаторов.

Драйвер локальных виртуальных дисков системы QEMU имеет параметр cache, задающий режим, в котором QEMU оперирует с файлом или устройством виртуального диска.

Четыре возможных значения этого параметра соответствуют четырем комбинациям флагов O_DIRECT и O_DSYNC системного вызова открытия файла: cache=none – O_DIRECT; cache=writethrough – O_DSYNC; cache=directsync – O_DIRECT и O_DSYNC; cache=writeback – без флагов. По стандарту POSIX [9] опция O_DIRECT запрещает кэширование данных в общесистемном файловом кэше, а опция O_DSYNC указывает на необходимость ожидания подтверждения от устройства накопителя каждой операции записи данных.

Режим cache=writeback не является надежным, так как он не гарантирует сохранности записанных данных при внезапном отключении питания. С другой стороны, полное отключение кэширования данных [1] (режимы cache=none и cache=directsync) не рекомендуется для всех накопителей данных кроме самых скоростных NVMe и NVRAM устройств, имеющих пропускные способности и задержки более близкие к характеристикам оперативной памяти, чем к дисковым устройствам. Таким образом, оптимальным режимом работы виртуальных дисков для локальных накопителей является режим кэширования данных со сквозной записью, cache=writethrough.

3. Измерения влияния файлового кэша на производительность ввода-вывода

Для оценки возможности ускорения дискового ввода-вывода в случае попадания данных в файловый кэш был выполнен набор тестов производительности, предоставляемых утилитой Flexible I/O (FIO) версии 3.25 [4]. Тесты выполнялись из виртуальной ОС по отношению к виртуальным дискам (файлам в формате RAW). Каждый тест выполнялся либо с виртуальным диском, полностью загруженным в файловый кэш и удерживаемым в нем командой vmtouch -l ..., либо наоборот, полностью вытесненным из кэша командой vmtouch -e ... перед запуском теста.

В таблицах 1 и 2 собраны результаты тестирования производительности виртуального диска на SSD и HDD устройствах в двух режимах:

- многопотоковый асинхронный режим, командой вида

```
fio -ioengine=libaio -rw=randrw -rwmixread=80 -bs=4k -direct=1 -iodepth=128 -numjobs=4 -sync=0 ...;
```

- однопотоковый синхронный режим, командой вида

```
fio -ioengine=libaio -rw=randrw -rwmixread=80 -bs=4k -direct=1 -iodepth=1 -numjobs=1 -sync=1 ....
```

Опции -rwmixread=80 -bs=4k задают режим тестирования чтения и записи блоков данных размеров 4096 байта в отношении 4 к 1 (80% — чтение, 20% — запись). В тестах SSD устройств (таблица 1) были задействованы два NVMe-накопителя данных модели Intel SSD Pro 6100p, объединенных в программный массив RAID1. В тестах HDD устройств (таблица 2) применялись четыре HDD-накопителя данных модели Seagate ES.3 ST2000NM0033, объединенных в программный массив RAID10. В обоих случаях виртуальным диском служил файл в формате RAW,

который был создан в режиме полной инициализации.

Тестовая утилита FIO запускалась из виртуальной ОС AlmaLinux 9. Для работы с виртуальным диском использовался драйвер диска VirtIO, наиболее производительный из драйверов виртуальных дисков в системе QEMU. Применение данного драйвера не добавляет накладных расходов эмуляции протокола обмена по шинам SCSI или SATA.

Основной результат тестов, приведенных в таблице 1, состоит в том, что использование файлового кэша позволяет ускорить в 2.36 раза скорость ввода-вывода SSD-накопителей в однопотоковом синхронном режиме. Из данных таблицы следует, что ускоряются не только операции чтения, что очевидно при кэшировании, но и операции записи. Приведенные в таблице значения задержки операций (последние три строки) означают полное время выполнения операции от ее запуска до получения подтверждения об окончании. Поскольку тесты FIO выполнялись с опцией -sync=1, которая форсировала вызов синхронизации данных после каждой записи, то получение подтверждения означало, что данные действительно записаны в постоянную память SSD-устройства. Из последней строки видно, что среднее время записи уменьшилось почти в 2 раза, тогда как минимальное время записи почти не изменилось. Это позволяет предположить, что при интенсивном использовании файлового кэша в очередь запросов SSD-устройства попадают преимущественно операции записи, и в этом режиме устройство реже останавливается для выполнения внутренних операций, таких, как сборка мусора и планирование равномерного использования ресурса перезаписи ячеек памяти.

Таблица 1. Результаты тестирования смешанной нагрузки ввода-вывода на SSD-накопителях

Тест	Кэш пуст	Кэш предзагружен	Ускорение
Многопотоковый асинхронный режим			
Чтение, операций в секунду	65200	73700	1.13
Запись, операций в секунду	16300	18400	1.13
Всего, операций в секунду	81500	92100	1.13
Минимальная задержка записи, µсек	2040	2080	1.02
Медианная задержка записи, µсек	16710	14350	1.16
Средняя задержка записи, µсек	17975	15760	1.14
Однопотоковый синхронный режим			
Чтение, операций в секунду	2090	4900	2.34
Запись, операций в секунду	510	1240	2.43
Всего, операций в секунду	2600	6140	2.36
Минимальная задержка записи, µсек	348	346	1.01
Медианная задержка записи, µсек	502	470	1.07
Средняя задержка записи, µсек	1010	535	1.89

Данный эффект противоречит популярным рекомендациям, согласно которым использовать файловый кэш по отношению к SSD-накопителям не имеет смысла и снижает производительность ввода-вывода. Опровержение этих рекомендаций открывает различные возможности оптимизации подсистемы ввода-вывода. Например, заметного ускорения можно ожидать при добавлении высокоскоростного NVMe- или PCIe-накопителя небольшого объема в качестве кэша системы логических томов LVM (основанного на модуле ядра dm-cache [5]). Данный кэш будет работать в режиме сквозной записи, что не снизит надежность всей системы, но при этом ускорится работа виртуальных дисков, расположенных на этом томе.

В случае многопотоковой асинхронной нагрузки наблюдаемое ускорение операций составляет всего 13%. В этих тестах SSD-устройство показывает значительно большую производительность, чем в однопотоковом тесте с синхронной записью. Но среднее время выполнения операции записи в этом тесте тоже выше более чем 15 раз, нежели в синхронном режиме. Асинхронная многопотоковая нагрузка — это тот тип нагрузки, для которого оптимизирована внутренняя программа SSD-устройства, поэтому в таком режиме устройство показывает наибольшую производительность. Кроме того, поскольку тестируемая модель SSD-накопителей, согласно ее спецификации, оптимизирована для операций чтения, то основное время в этом тесте расходовалось на запись данных, и использование файлового кэша не дало заметного прироста производительности.

Таблица 2. Результаты тестирования смешанной нагрузки ввода-вывода на HDD-накопителях

Тест	Кэш пуст	Кэш предзагружен	Ускорение
Многопотоковый асинхронный режим			
Чтение, операций в секунду	450	852	1.89
Запись, операций в секунду	113	211	1.87
Всего, операций в секунду	563	1063	1.89
Минимальная задержка записи, µсек	261940	48200	5.43
Медианная задержка записи, µсек	1166015	677330	1.72
Средняя задержка записи, µсек	1145420	624950	1.83
Однопотоковый синхронный режим			
Чтение, операций в секунду	53	145	2.74
Запись, операций в секунду	13	35	2.69
Всего, операций в секунду	66	180	2.73
Минимальная задержка записи, µсек	6745	4470	1.51
Медианная задержка записи, µсек	37770	21365	1.77
Средняя задержка записи, µсек	40108	26930	1.52

Из таблицы 2 видно, что для HDD-устройств наблюдается 2.7x и 1.9x кратное ускорение для синхронной и асинхронной работы, соответственно. Но причины ускорения от использования кэширования в случае HDD отличаются от случая SSD. Скорость доступа к данным у механических дисковых накопителей значительно ниже, чем у твердотельных SSD-накопителей и в основном определяется скоростью точного позиционирования блока магнитных головок к дорожке с данными. Следовательно, операции чтения или записи произвольного блока данных в механическом дисковом накопителе должны иметь одинаковые средние времена выполнения (если не рассматривать устройства с «черепичной» записью — Shingled Magnetic Recording, SMR), тогда как запись данных SSD-устройством выполняется медленнее чтения. Также можно считать, что в тестах с пустым кэшем все операции чтения выполняются непосредственно из накопителя (так как диск большой и случайное чтение за время теста имеет незначительное количество повторно прочтенных блоков), а в тестах с полностью загруженным кэшем все операции чтения выполняются из кэша. Поэтому в асинхронном многопотоковом режиме можно было бы ожидать 5-кратного увеличения количества операций записи, так как все операции чтения выполняются из кэша, а их в смешанной нагрузке 80%. Но наблюдаемое увеличение меньше 5-кратного — от 113 до 211 операций в секунду. Это расхождение частично объясняется тем, что в массиве RAID10 операции чтения могут выполняться параллельно, а операции записи — нет. Если учесть эту поправку, то увеличение скорости записи должно быть не 5-кратным, а 3-кратным. Но поскольку 211 операций записи в секунду приблизительно соответствуют оценке пиковой производительности произвольного доступа для HDD-накопителя, вопрос скорее в том, почему в первой колонке (тест с пустым кэшем) удалось выполнить 563 операции в секунду. Вероятно, в режиме смешанной асинхронной нагрузки жесткий диск имеет больше возможностей по оптимизации плана выполнения запросов и более успешно использует внутреннюю кэш-память. В тесте синхронного доступа небольшое количество операций записи (35 операций/сек) совпадало с результатами аналогичного теста на сервере виртуализации. HDD-накопители со скоростью вращения 7200 об/мин выполняют одиночную операцию записи за среднее время 10-15мсек, что соответствует 67-100 операциям/сек. Но в режиме RAID10 требуется ожидать подтверждения записи двух копий данных от двух устройств, что увеличивает среднее время операции. Кроме того, тестируемые HDD-накопители находились в эксплуатации более 8 лет, и их высокий износ мог привести к увеличению времени точного позиционирования блока головок.

4. Программа динамической оптимизации кэширования виртуальных дисков

Приведенные выше результаты тестов демонстрируют возможность ускорения ввода-вывода за счет применения файлового кэша для виртуальных дисков в случае своевременной загрузки необходимых данных.

Кэширование данных открытых файлов выполняется в рамках работы более общего механизма управления использованием страниц памяти. Традиционно ядро Linux создает два списка страниц памяти для алгоритма удержа-

ния в ОЗУ наиболее часто используемых страниц (least-recently-used, LRU). Страницы, к которым недавно осуществлялся доступ, помещаются в начало списка «активных» страниц. Страницы из хвоста этого списка удаляются, если к ним давно не обращались, и помещаются в начало списка «неактивных» страниц. Когда какой-либо процесс повторно обращается к «неактивной» странице, она помещается обратно в список «активных» страниц. В случае дефицита памяти ядро удаляет первыми страницы из хвоста списка «неактивных» страниц. Если список «неактивных» страниц становится короче половины длины списка «активных» страниц, то выполняется перераспределение страниц этих списков для сохранения указанного соотношения длин списков. Следует отметить, что для определения факта использования страниц памяти применяется бит Accessed в структуре Page Table Entry (PTE), который автоматически выставляется устройством управления памятью (MMU).

Таким образом, алгоритм LRU приближенно решает задачу оценки вероятности будущего доступа к ранее использованной странице памяти, почти не задействуя для этого вычислительные ресурсы. В современных версиях ядра Linux картина усложняется, если имеется несколько процессоров с неоднородной архитектурой памяти (non-uniform-memory-access, NUMA) или применяется механизм изоляции ресурсов cgroups — тогда вышеописанные списки создаются для каждого процессора и каждой группы процессов cgroups. Подробное описание механизма управления памятью, список соответствующих ему параметров и методов диагностики содержится в документации ядра Linux [6].

Отметим следующие недостатки общесистемного файлового кэша:

- он универсален — у пользователя нет возможности выделить больше памяти для определенных файлов;
- он имеет небольшую гранулярность, равную размеру страницы памяти (4096 байт), при этом нет возможности задать больший размер блоков, чтобы данные загружались «по ассоциации».

Частично решить задачу распределения приоритетов в использовании оперативной памяти позволяет современный механизм изоляции ресурсов cgroups v2, имеющий параметры memory.low и memory.high для каждой группы процессов. Эти параметры влияют на принятие решений об увеличении или уменьшении количества страниц памяти, необходимых данной группе процессов. В среде виртуализации Libvirt/QEMU/KVM каждая виртуальная ОС помещается в отдельную группу cgroup, поэтому такой способ оптимизации вполне возможен. Однако, этот метод требует точной оценки потребности в оперативной памяти всех виртуальных ОС на сервере и не гарантирует успеха, поскольку указываются рекомендуемые размеры всей памяти для cgroup, что не обязательно приведет к более активному кэшированию содержимого виртуального диска, скорость работы которого требуется увеличить.

Еще одним параметром, косвенно влияющим на работу файлового кэша, является размер окна опережающего чтения (read-ahead size), который можно задать либо для блочного устройства на сервере виртуализации, либо внутри виртуальной ОС. Но увеличение этого параметра не всегда приводит к желаемому результату, так как оно повышает количество прочитываемых и удерживаемых в кэше данных, не все из которых оказываются нужны.

С другой стороны, задача приоритетного удержания часто используемых данных виртуальных дисков допускает и более простое решение. Так, ядро Linux имеет системный вызов mincore, который возвращает список находящихся в файловом кэше блоков указанного файла. К этому системному вызову обращается утилита vmtouch [7], которая в зависимости от указанных параметров либо сообщает пользователю, какая часть интересующего его файла находится в кэше, либо загружает и удерживает весь этот файл или некоторую его часть в кэше. Попытки применить утилиту vmtouch для удержания в кэше всего виртуального диска или некоторой его части обычно не реалистичны, так как часть данных диска, которая будет интенсивно использоваться, заранее неизвестна, а размер всего диска, как правило, превосходит размер оперативной памяти сервера. Исключением служит ситуация, когда для ускорения работы некоторого приложения в виртуальной ОС создается отдельный виртуальный диск небольшого размера, а затем файл этого диска полностью удерживается в кэш-памяти утилитой vmtouch. Развитием утилиты vmtouch является утилита vmprobe [8], которая также имеет функции сохранения «моментального снимка» состояния кэш-памяти применительно к указанному файлу и последующего восстановления состояний кэш-памяти из сохраненных таким образом «снимков». Эти функции утилиты vmprobe позволяют, например, зафиксировать определенное состояние сервера базы данных, часто называемое «прогретым» состоянием, и в дальнейшем при старте сервера быстрее достигать этого состояния, форсируя загрузку необходимых данных в файловый кэш. Тем не менее, утилиты vmtouch и vmprobe не проводят динамического

анализа использования файлового кэша, и поэтому не способны перенастраивать множество загруженных страниц данных в реальном времени.

Для более гибкого управления кэшированием виртуальных дисков авторами данной статьи была разработана утилита vmdisktouch, которая является развитием утилиты vmtouch и выполняет следующие действия:

- периодически делает системный вызов mincore и составляет карту загруженных в файловый кэш блоков виртуального диска;
- находит области диска, где недавно выполнялось много операций чтения и записи, и помечает их как области, которые имеет смысл удерживать в кэше;
- удерживает в кэше данные выбранных областей виртуального диска;
- выводит «тепловую карту» файлового кэша с обозначением всех удерживаемых на данный момент в кэше блоков (в том числе недавно перезаписанных) и областей.

Утилита написана на языке Python с использованием библиотек NumPy и SciPy для обработки массивов данных, библиотек mmap и fincore для работы в виртуальной памятью системы Linux, и библиотеки Pillow для вывода графических PNG файлов. Функции данной утилиты дают администратору сервера виртуализации возможность наблюдать за использованием файлового кэша по отношению к виртуальным дискам и выбирать оптимальный режим динамической загрузки часто используемых областей виртуального каждого диска.

5. Алгоритм работы утилиты vmdisktouch

Чтобы определить часто используемые области виртуального диска, утилита vmdisktouch применяет к битовой карте блоков файлового кэша операцию сглаживания (фильтр Гаусса) с заданной дисперсией и амплитудой. Получившаяся функция обновляет более медленно меняющуюся функцию, в которой накапливается экспоненциально затухающее среднее значение. Последнюю функцию можно интерпретировать как функцию интенсивности доступа или «тепловую карту» использования виртуального диска. Она представляет собой приближение к функции плотности вероятности событий ввода-вывода в массиве блоков данных виртуального диска. Далее в массиве этой функции выделяются сегменты блоков данных, в которых значения превышают некоторый порог. Эти сегменты затем становятся областями, которые будут удерживаться в файловом кэше.

Поскольку в результате работы этого алгоритма все блоки найденных областей попадают в кэш, их биты принудительно обнуляются перед применением операции сглаживания. Это выключает положительную обратную связь, которая иначе привела бы к монотонному распространению этих областей на все пространство блоков диска. Кроме того, обнуление этих битов приводит к постепенному уменьшению функции в данном регионе, поэтому удерживаемая в кэше область данных через некоторое время перестает удерживаться, но только если в этой области не было большого числа операций записи данных, которые также увеличивают значения функции (см. ниже). Такое динамическое переопределение множества областей, удерживаемых в кэш-памяти, помогает находить регионы диска, в которых наблюдается высокая интенсивность операций ввода-вывода.

Для обнаружения операций записи, выполняемых виртуальной ОС, утилита периодически прочитывает и вычисляет контрольные суммы блоков данных удерживаемых в файловом кэше областей. Перезапись данных виртуального диска сопровождается обновлением содержимого файлового кэша, поэтому утилита обнаружит измененные контрольные суммы и пометит блоки как перезаписанные. Утилита не может обнаруживать изменение в остальных блоках данных, поскольку само событие их чтения повлияло бы на работу файлового кэша и заставило продолжать хранить эти блоки в кэше, часто безосновательно.

Согласно логике данного алгоритма, утилита в начале своей работы наблюдает за картой загруженных в кэш блоков данных и находит области с высокой плотностью таких блоков. Затем наиболее значимые области выбираются для их полной загрузки в файловый кэш. Далее в процессе работы поступает информация о перезаписываемых блоках и добавляет информацию об областях, где данные не только читаются, но и перезаписываются. Эти области получают наибольший приоритет и дольше удерживаются в файловом кэше.

Утилита vmdisktouch принимает следующие параметры:

- имя файла виртуального диска;
- максимальный размер данных для загрузки в файловый кэш;
- минимальное значение свободной общесистемной памяти, при котором утилите разрешено загружать в кэш страницы памяти;
- длительность цикла работы утилиты;
- параметры амплитуды и ширины Гауссова фильтра для карты блоков файлового кэша и отдельно для карты перезаписанных блоков;

• коэффициент затухания для экспоненциального среднего значения функции интенсивности доступа;

• порог значения функции интенсивности доступа для определения областей, загружаемых в файловый кэш;

• имена PNG-файлов, в которые выводятся два изображения: «тепловой карты» для функции интенсивности и изображение, где точно обозначены блоки данных, загруженные в файловый кэш;

• параметры вывода изображения: число блоков на пиксел, соотношение сторон, и масштаб изображения.

Обязательными параметрами являются только первые два, для остальных же утилита выбирает значения, близкие к оптимальным и выбранные на основании опыта ее применения. При обновлении множества удерживаемых в кэше областей виртуального диска утилита использует приоритетную очередь, в которой значение приоритета — сумма функции интенсивности доступа. Это обеспечивает оптимальность выбора таких областей, если загрузить все найденные области не позволяет ограничение на общий размер (второй параметр утилиты).

Настройка остальных параметров алгоритма утилиты устанавливает режим работы утилиты. Например, с помощью амплитуды фильтра для перезаписанных блоков можно задать более высокий приоритет тем областям данных виртуального диска, где часто выполняются операции записи. Коэффициент затухания функции и длительность цикла определяют скорость перенастройки загруженных в кэш областей данных.

За работой утилиты удобно наблюдать, открыв PNG-файл изображения «тепловой карты» виртуального диска. Многие программы просмотра изображений в системе Linux отслеживают изменение содержимого файла и обновляют картинку, что дает эффект отображения в режиме реального времени. Для отрисовки «тепловой карты» используется цветовая палитра, обозначающая тип блоков данных: синий цвет соответствует блокам, загруженным по инициативе утилиты vmdisktouch, зеленый цвет означает загрузку по инициативе виртуальной ОС, красным цветом отмечены недавно перезаписанные блоки данных.

6. Заключение

Разработанная авторами утилита vmdisktouch позволяет более интенсивно кэшировать данные виртуального диска и выбирать для этого преимущественно те области диска, которые участвуют одновременно в чтении и записи данных. Оказывается, что такое кэширование увеличивает также скорость операций записи SSD-накопителей, особенно в синхронном режиме. Использование данной утилиты не нарушает функционирование виртуальных ОС, поскольку ее работа заключается только в опросе состояния файлового кэша системы Linux и чтении данных виртуальных дисков.

Недостатком алгоритма работы утилиты vmdisktouch является то, что одно лишь наблюдение за блоками виртуального диска, находящимися в файловом кэше, делает затруднительным определение причины попадания этих блоков в кэш. Алгоритм должен запоминать блоки, загрузка которых была осуществлена по его инициативе, и исключать их из дальнейшего динамического анализа, иначе области распространятся постепенно на весь виртуальный диск. При этом теряется информация о том, понадобились ли некоторые из этих блоков, т.е. производились ли операции чтения этих блоков данных со стороны виртуальной ОС. По этой же причине не удается детектировать события записи блоков вне областей, уже удерживаемых в памяти. Если бы система виртуализации позволяла наблюдать за событиями и чтения и записи блоков данных виртуального диска, то можно было бы более точно определять области данных, рекомендованные для удержания в кэше.

Дальнейшим развитием функциональности утилиты vmdisktouch может быть реализация следующих возможностей:

• отслеживание перезаписанных блоков данных с помощью карты обновленных блоков виртуального диска (функция block-dirty-bitmap, добавленная в QEMU v.2.4). Этот метод позволит получать список всех выполненных операций записи данных, что увеличит точность работы утилиты и сэкономит процессорный ресурс;

• временной анализ событий ввода-вывода с обнаружением закономерностей и предсказанием будущих событий для опережающей загрузки блоков данных в файловый кэш.

Работа выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0001 «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (1023032100070-3-1.2.1).

A Dynamic Analysis and Optimization of I/O in the GNU Linux/QEMU/KVM Virtualization Environment

A.B. Betelin, G.A. Prilipko, A.G. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. In this paper we describe results of virtual disk I/O performance tests in GNU Linux/QEMU/KVM environment and propose optimization methods that improve use of the system page cache. The authors have developed a utility for dynamic analysis and optimization of virtual disk usage. Testing has shown that this utility can provide more than 2x acceleration for mixed I/O workloads on SSDs.

Keywords: virtualization, I/O, QEMU, KVM, Linux, vmtouch, vmprobe, mincore

Литература

1. Сайт "QEMU, the FAST! processor emulator". https://www.qemu.org (дата обращения 06.03.2024).

2. Сайт проекта KVM. https://www.linux-kvm.org (дата обращения 06.03.2024).

3. А.Б. Бетелин, И.Б. Егорычев, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. Настройка и оптимизация системы ввода-вывода в среде виртуализации GNU Linux/QEMU/KVM/Libvirt. «Труды НИИСИ РАН», т.9 (2019), № 5, 119–129.

4. Сайт документации утилиты Flexible I/O tester (FIO). https://fio.readthedocs.io/en/latest (дата обращения 06.03.2024).

5. Сайт документации ядра Linux, раздел "Device Mapper. Cache". https://www.kernel.org/doc/Documentation/device-mapper/cache.txt (дата обращения 06.03.2024).

6. Сайт документации ядра Linux, раздел "Memory Management". https://www.kernel.org/doc/html/latest/admin-guide/mm (дата обращения 06.03.2024).

7. Сайт утилиты vmtouch. https://hoytech.com/vmtouch (дата обращения 06.03.2024).

8. Сайт утилиты vmprobe. https://vmprobe.com/intro (дата обращения 06.03.2024).

9. Стандарт POSIX.1-2017. The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017.