Отчёт по преддипломной практике. Выявление средствами профилирования фрагментов кода QEMU-KVM, критических для производительности

Черемнов А.В.

1 Настройка виртуальных машин

Хостовая операционная система —Ubuntu 16.04, гостевая —аналогично. Версия ядра 4.15.0.70-generic.

Для корректной работы профилировщика в гостевой виртуальной машине необходимо включить виртуализацию PMU. Тем самым, виртуализируются процессорные счётчики, за счёт которых и работает perf. Для виртуализации счётчиков используется аппаратная поддержка виртуализации, и уже существующий PMU используется в том числе и для подсчёта внутри гостя.

Информация о символах ядра и подключённых модулях гостевой операционной системы доступна из хостовой операционной системы.

В дальнейшем, если не указано обратное, считается, что в гостевой ОС работают только фоновые процессы, пользователь бездействует в терминале.

2 Perf и его особенности

perf - средство для профилирования в Linux, входящее в число стандартных инструментов (linux-tools). Для профилирования использует аппаратные и программные счётчики.

2.1 Вид отчёта perf

Во всех рассмотренных ниже примерах отчёт perf сформирован с поддержкой графов вызовов. Это позволяет произвести более точный анализ критически важных областей.

Первый и второй столбец отчёта — children и self соответственно. Children — суммарные накладные расходы всех детей данного узла графа вызовов, self -именно этого узла графа. Последний столбец обозначает функцию или область памяти, в которую попали семплы.

2.2 Точность perf

Однако следует учитывать, что perf не выдаёт точных результатов: точность может ухудшаться не только из-за частоты семплирования, но и мультиплексирования. Все оценки приблизительны.

Семпл - значение счётчика команд в данный момент времени. Семплирование - периодичный процесс измерения семплов. Накладные расходы — это процентное соотношение семплов, попавших в данную область. Семплирование может основываться на частоте или на количестве событий. В первом случае заборы семплов производятся каждый фиксированный промежуток времени, во втором — при превышении счётчиком событий заданной величины.

При семплировании на основе частоты может возникнуть проблема с циклом той же частотой, что и семплирование. К примеру, есть функция, которая вызывается в таком цикле. В таком случае, при семплировании данной функции может не оказаться никогда семплов, хотя она может быть значима. Или, наоборот, незначимая функция может быть во множестве семплов. Поэтому необходимо анализировать несколько вариантов частот при профилировании. Другой, менее надёжный вариант — применить эвристику, к примеру, сделать частоту нечётной, исходя из предположения, что у большой части циклов частота чётная.

Мультиплексирование применяется тогда, когда число событий, которые надо подсчитать, превышает число аппаратных счётчиков. Тогда в каждый конкретный момент измеряются не все события, и с определённой частотой измеряемые события меняются. Затем, по окончанию профилирования семплы данного события масштабируются, компенсируя время, пока они не измерялись. Таким образом, чем больше событий рассматриваются в рамках одного запуска профилировщика, тем менее точным будет результат.

2.3 Отслеживание событий гостя в perf

Используя perf stat, можно профилировать количество инструкций, выполнившихся на уровне гостя и события, возникшие в результате выполнения данных инструкций.

Этого можно добиться, добавив модификатор соответствующего уровня. Нам интересны модификаторы G — уровень ядра гостевой виртуальной машины, u — уровень гостевой операционной системы, H - уровень гипервизора. Результаты относительно конкретной виртуальной машины можно получить профилированием на уровне гостевой операционной системы и определённого процесса KVM.

Рассмотрим на примере: cache-misses. Напомним, что профилирование из-под гостя доступно только в случае включения виртуализация программных счётчиков. На основе полученных данных о госте можно составить представление о наиболее затратных функциях уровня гостя. Если соотнести это с критическими фрагментами кода QEMU/KVM, можно получить информацию о характере зависимости между ними. Так, если в госте за-

тратные операции связаны с CPU, а в хосте - с I/O, то можно сделать вывод, что выполнение I/O операций в QEMU-KVM крайне затратный процесс.

3 Проблемы

При выполнении задания по преддипломной практике возникла проблема с профилированием гостя вместе с монитором виртуальных машин(KVM).

Целью профилирования было одновременное отслеживание гостевой операционной системы и хостовой операционной системы. Однако в некоторых случаях граф вызовов гостя отсутствует, в некоторых - присутствует. Проблема была разрешена после настройки виртуализации PMU.

Perf работал некорректно на perf test — Dwarf unwind failed. Проблема была решена обновлением perf.

4 Выявление критических фрагментов кода QEMU-KVM

4.1 Анализ

События	Наибольшие затраты
Инструкции	ioctl
Промахи в кэше	ppoll
Общее время	Внешние прерывания ioctl
Процессорные циклы	Вход/выход из монитора

Наибольший интерес представляют события, происходящие в гипервизоре. Именно его работу нам надо проанализировать. Модификатором можно отследить все инструкции, выполнившиеся на уровне гипервизора. Другой вариант - узнать pid, соответствующий данной конкретной виртуальной машине. Профилирование этого процесса позволит отследить накладные расходы гипервизора в контексте данной виртуальной машины.

Во всех нижерассмотренных примерах рассмотрим распределение нагрузки на гипервизор в случае бездействия гостевой ОС (и работы только фоновых процессов).

Как следует из сгенерированного отчёта, наибольшее количество инструкций затрачено на вход в виртуальную ОС и виртуализацию ioctl. Системный вызов ioctl осуществляется из функции, контролирующей виртуальную файловую систему (VFS). Сама ioctl отвечает за операции ввода-вывода с устройствами.Однако наибольшие одномоментные затраты приходятся на memcmp, осуществляемую в ksmd(kernel same-page merging). Кроме того, в KVM 3 процента инструкций занимают инструкции SCTP timer. Однако ни ksmd, ни SCTP timer не так затратны по времени, как следует из примеров ниже.

Во втором примере проанализируем промахи в кэше. За наибольшее количество промахов в кэше ответственно `sys_ppoll`. Наибольшие собственные накладные расходы — `copy_user_enhanced_fast_string` (обмен сообщения TCP) и `fget`. Промахи в кэше из-за вызовов ioctl занимают примерно 2 процента от всех промахов. (Рассматривался кэш первого уровня)

Все `kvm_page_fault` обрабатываются по одному и тому же сценарию: до `handle_ept_violation`. Эта функция обрабатывает нарушение Extended Page Tables - механизма Intel для эффективной виртуализации памяти. Согласно нижеприведённым данным о процессорных циклах, такая функция не входит в число наиболее затратных по производительности, несмотря на то, что требует входа в систему виртуализации. Следовательно, обработка страничных нарушений KVM не занимает существенного процессорного времени.

`cpu-clock` - доходящие до 88 процентов затраты на выполнение операций с vfs (ioctl). 70 процентов всего времени уходит на обработку внешних прерываний. Однако стоит учитывать, что `cpu-clock` измеряет время с помощью встроенных в Linux инструментов.

С другой стороны, `cycles` показывает схожие и одновременно существенно отличающиеся результаты. 70 процентов всех циклов — в операциях с vfs(ioctl). Однако на обработку внешних прерываний уходит всего 8 процентов.

Причиной таких различий является особенности события cycles - измеряется число только активных, не приостановленных циклов процессора. А в момент ожидания внешнего прерывания процесс приостанавливается.

Обычных страничных нарушений внутри kvm не происходит, исключительно `kvm_page_fault`.

Наибольшее число циклов шины занимают все те же операции с файловой системой и ioctl. Однако, по сравнению с обычными циклами, операции polling(синхронный IO) занимают больше в процентном соотношении.

4.2 Возможности улучшения произведённого анализа

Для дополнительного анализа наиболее затратных функций возможно добавить на них probe. С другой стороны, тогда возникает необходимость в debuginfo.

Особый интерес представляют вопросы обработки нарушений страниц в условиях KVM. Следует обратить внимание на анализ трассируемого события `kvm_page_fault`. При использование perf script можно наблюдать получившуюся трассу. Все страничные нарушения kvm происходили в `handle_ept_violation`

5 Выводы

Наиболее критическим участком для KVM в случае бездействующей гостевой системы (работают только фоновые процессы и терминал) является работа с I/O, конкретнее - ioctl. Как необходимость входа и выхода из

монитора, так и само ожидание внешних прерываний вносят наиболее существенный вклад. MMIO и связанные с ним необходимость переключения контекстов между гостевой операционной системой и монитором.

По промахам в кэше критична ppoll, то есть синхронное I/O чаще вызывает промахи в кэше, чем асинхронное. [1]

6 Список литературы

[1] Jiaqing Du, Nipun Sehrawat и Willy Zwaenepoel. “Performance Profiling in a Virtualized Environment”. в: (2010).