Масштабирование сетевого стека Linux

По материалам ядра Linux [1] [2]

Введение

В этом документе описан набор дополнительных методов для улучшения распараллеливания и повышения производительности сетевого стека Linux в многопроцессорных системах.

- RSS: Receive Side Scaling - масштабирование на приёмной стороне.
- RPS: Receive Packet Steering - распределение принимаемых пакетов (начиная с ядра 2.6.35).
- RFS: Receive Flow Steering - распределение принимаемых потоков (начиная с ядра 2.6.35).
- Accelerated Receive Flow Steering - ускоренное распределение принимаемых потоков (начиная с ядра 2.6.35).
- XPS: Transmit Packet Steering - распределение передаваемых пакетов (начиная с ядра 2.6.38).

RSS

Современные сетевые адаптеры (NIC) поддерживают несколько очередей дескрипторов приёма и передачи (множество очередей - multi-queue). На приёме NIC может помещать пакеты в разные очереди для распределения нагрузки между процессорами (CPU). Сетевой адаптер распределяет пакеты, применяя к каждому из них фильтр, относящий пакет к одному из небольшого числа логических потоков. Пакеты каждого потока направляются в свою приёмную очередь, а эти очереди могут обрабатываться разными CPU. Этот механизм обычно называют масштабированием не приёмной стороне (Receive-side Scaling или RSS). Целью RSS и других методов масштабирования является однородное повышение производительности. Распределение между множеством очередей может также применяться для приоритизации трафика, но это не является основной задачей описываемых методов.

Применяемым в RSS фильтром обычно является функция свёртки (хэш - hash) для заголовков сетевого и/или транспортного уровня, например, хэш-значение для полей адресов IP и портов TCP в заголовке пакета. В наиболее распространённой аппаратной реализации RSS применяется таблица перенаправления со 128 записями, каждая из которых содержит номер очереди. Приёмная очередь для пакета определяется маскированием 7 младших битов рассчитанного для пакета хэш-значения (обычно Toeplitz-хэш [3]), задающим ключ для таблицы перенаправления, и считыванием соответствующей записи из таблицы.

Некоторые современные NIC позволяют направлять пакеты в разные очереди на основе программируемых фильтров. Например, пакеты для web-сервера на порту TCP 80 могут направляться в его собственную приёмную очередь. Такие фильтры (n-tuple) можно настраивать с помощью утилиты ethtool с командой --config-ntuple.

Конфигурация RSS

Драйверы NIC с поддержкой нескольких очередей обычно предоставляют параметр модуля ядра, задающий число аппаратных очередей. Например, в драйвере bnx2x это параметр num_queues. Типичная конфигурация RSS включает 1 приёмную очередь для каждого CPU, если устройство поддерживает достаточно очередей, или хотя бы 1 очередь на область (domain) памяти - набор CPU, использующих общий уровень памяти (L1, L2, узел NUMA и т. п.).

Таблица перенаправления устройства RSS, которая определяет очередь по маске хэш-значения, обычно программируется при инициализации драйвера. По умолчанию пакеты равномерно распределяются по очередям, но таблицу перенаправления можно извлечь и изменить во время работы с помощью утилиты ethtool (команды --show-rxfh-indir и --set-rxfh-indir). Изменение таблицы позволяет задать для очередей разный вес. Ниже приведён пример принятой по умолчанию таблицы для сетевого адаптера I210 Gigabit Network Connection (драйвер igh).

```
# ethtool --show-rxfh-indir enp4s0

RX flow hash indirection table for enp4s0 with 4 RX ring(s):
   0:      0     0     0     0     0     0     0     0
   8:      0     0     0     0     0     0     0     0
  16:      0     0     0     0     0     0     0     0
  24:      0     0     0     0     0     0     0     0
  32:      1     1     1     1     1     1     1     1
  40:      1     1     1     1     1     1     1     1
  48:      1     1     1     1     1     1     1     1
  56:      1     1     1     1     1     1     1     1
  64:      2     2     2     2     2     2     2     2
  72:      2     2     2     2     2     2     2     2
  80:      2     2     2     2     2     2     2     2
  88:      2     2     2     2     2     2     2     2
  96:      3     3     3     3     3     3     3     3
104:      3     3     3     3     3     3     3     3
112:      3     3     3     3     3     3     3     3
120:      3     3     3     3     3     3     3     3
RSS hash key:
Operation not supported
RSS hash function:
  toeplitz: on
  xor: off
  crc32: off
```

Конфигурация RSS IRQ

С каждой приёмной очередью связано своё значение прерывания (IRQ), которое NIC использует для уведомления CPU при поступлении новых пакетов в данную очередь. Сигнальным путём для устройств PCIe служат указываемые сообщениями прерывания (message signaled interrupt или MSI-X), которые могут маршрутизироваться конкретному CPU. Активное сопоставление очередей с IRQ можно узнать из файла /proc/interrupts. По умолчанию IRQ может обрабатываться любым CPU. Поскольку обработка прерывания, связанного с приёмом, включает достаточно большую часть обработки пакета, целесообразно распределять приёмные прерывания между CPU. Настройка близости (affinity)

для IRQ описана в приложении «Близость SMP IRQ». В некоторых системах используется демон irqbalance, который динамически оптимизирует назначение IRQ и может менять заданные вручную настройки.

Рекомендуемая конфигурация

RSS следует включать, когда задержки играют важную роль или обработка прерываний является узким местом. Распределение нагрузки между CPU сокращает размер очереди. Для сетей с малыми задержками оптимально задавать число очередей равным числу CPU в системе или устанавливать максимальное для NIC значение, если оно меньше числа процессоров. Наиболее эффективной высокоскоростной конфигурацией является конфигурация с наименьшим числом приёмных очередей, в которой ни одна из этих очередей не переполняется из полной загрузки CPU, поскольку в принятом по умолчанию режиме с включённым слиянием (coalescing) прерываний, суммарное число прерываний (а значит, и объём работы) растёт с каждой добавочной очередью.

Нагрузку на каждый процессор можно наблюдать с помощью утилиты mpstat, однако на процессорах с гиперпотоками (hyperthreading или HT), каждый такой поток представляется отдельным CPU. В части обработки прерываний исходные тесты HT не показали каких-либо преимуществ, поэтому следует ограничивать число очередей числом процессорных ядер в системе.

RPS

Распределение (направление) принимаемых пакетов (Receive Packet Steering или RPS) логически является программной реализацией RSS. Программная реализация ведёт к более позднему вызову в пути данных. RSS выбирает очередь и, следовательно, CPU где будет работать обработчик аппаратных прерывания, а RPS выбирает CPU для протокольной обработки над обработчиком прерываний. Для этого пакет помещается в очередь невыполненных заданий (backlog queue) выбранного CPU и активирует CPU для выполнения работы. RPS имеет некоторые преимущества по сравнению с RSS:

1. возможность применения с любым NIC;
2. простота добавления фильтров для хэширования новых протоколов;
3. отсутствие роста частоты аппаратных прерываний (однако добавляются межпроцессорные прерывания[1]).

RPS вызывается в нижней половине обработчика приёмных прерываний, когда драйвер отправляет пакет вверх по сетевому стеку с помощью вызова netif_rx() или netif_receive_skb(). Это вызывает функцию get_rps_cpu(), выбирающую очередь, которой следует обрабатывать пакет.

Первым шагом при определении целевого CPU для RPS является расчёт хэш-значения для потока по адресам и/или порта (2-tuple или 4-tuple в зависимости от протокола), которое связывается с пакетом. Хэш-значение предоставляется оборудованием или рассчитывается сетевым стеком. Поддерживающее хэширование оборудование может передать хэш в приёмном дескрипторе для пакета и обычно это будет то же значение, которое применяется для RSS (например, Toeplitz). Хэш сохраняется в поле skb->hash и может использоваться в стеке в качестве хэш-значения для потока.

Каждая аппаратная очередь приёма имеет список связанных CPU, которым RPS может передавать пакеты для обработки. Для каждого принятого пакета вычисляется индекс списка на основе хэш-значения по модулю в соответствии с размером списка. Указанный индексом CPU выбирается для обработки пакета и тот помещается в конец очереди невыполненных заданий CPU. В конце нижней половины процедуры обработки межпроцессорные прерывания IPI передаются всем CPU, для которых имеются пакеты в очереди невыполненных заданий. IPI активирует исполнение ожидающей работы соответствующими CPU и все пакеты из очередей обрабатываются в сетевом стеке.

Конфигурация RPS

Для работы RPS требуется ядро, собранное с опцией CONFIG_RPS (включена по умолчанию для SMP). Однако по умолчанию RPS не включается и требуется явное включение. Список CPU, которым RPS может пересылать трафик для каждой приёмной очереди задаётся файлом в /sys/class/net/<dev>/queues/rx-<n>/rps_cpus, где <dev> - имя сетевого интерфейса, <n> - номер очереди. Этот файл содержит битовую карту CPU. Распределение RPS отключено при нулевом значении (принято по умолчанию) и в этом случае пакеты обрабатываются прерывающим CPU. Описание битовых карт привязки CPU дано в Приложении «Близость SMP IRQ».

Рекомендуемая конфигурация

Для устройства с одной очередью в типичной конфигурации RPS файл rps_cpus указывает CPU в одном домене памяти с прерывающим CPU. Если локальность NUMA не является проблемой, могут быть указаны все CPU в системе. При высокой частоте прерываний может оказаться разумным исключение прерывающего CPU из карты, поскольку это ядро уже выполняет большой объем работы.

Если для систем с несколькими очередями конфигурация RSS задаёт отображение аппаратной очереди приёма на каждый CPU, распределение RPS становится избыточным и ненужным. При числе аппаратных очередей меньше числа CPU, распределение RPS может оказаться полезным, если rps_cpus для каждой очереди использует тот же домен памяти, что и прерывающий CPU для этой очереди.

Ограничения для потоков в RPS

RPS распределяет обработку приёма в ядре между CPU без нарушения порядка пакетов. При отправке всех пакетов одного потока на одно ядро (CPU) возникает дисбаланс нагрузки на CPU, если скорости потоков различаются. Крайним случаем является доминирование в трафике одного потока. Такое поведение, особенно на серверах с множеством одновременных подключений говорит о некорректной настройке или наличии атаки на службу (Denial of Service или DoS) с использованием обманных адресов источников.

Ограничение для потоков (Flow Limit) является необязательной функцией RPS, которая отдаёт приоритет небольшим потокам при конкуренции за CPU путём отбрасывания пакетов в больших потоках немного раньше, чем в мелких. Эта функция применяется лишь в случаях, когда целевой CPU для RPS или RFS близок к насыщению. Когда очередь входящих пакетов CPU достигает половины своего максимального размера (sysctl net.core.netdev_max_backlog), ядро запускает счётчики пакетов по потокам для последних 256 пакетов. Если при поступлении нового пакета поток

[1] Inter-processor interrupt или IPI.

превышает установленное ограничение (по умолчанию половина прибывающих пакетов), этот пакет отбрасывается. Пакеты из других потоков по-прежнему отбрасываются лишь при достижении netdev_max_backlog. Пока размер входной очереди ниже заданного порога, пакеты не отбрасываются, поэтому ограничение потоков не разрывает соединения сразу и связность сохраняется даже для больших потоков.

Интерфейс

Возможность ограничения для потоков по умолчанию включена в ядро (CONFIG_NET_FLOW_LIMIT), но само ограничение по умолчанию выключено. Ограничения задаются независимо для каждого CPU (чтобы избежать конфликтов блокировки и кэширования) и включаются для CPU установкой соответствующего бита маски в sysctl net.core.flow_limit_cpu_bitmap. Используется тот же формат маски, как в rps_cpus (см. выше) при вызове из procfs

`/proc/sys/net/core/flow_limit_cpu_bitmap`

Скорость на уровне потока рассчитывается путём хэширования каждого пакета в ячейку (bucket) хэш-таблицы и инкрементирования счётчика по ячейкам. Применяется та же функция хэширования, что и для выбора CPU в RPS, но число ячеек может быть значительно больше числа CPU, поэтому для ограничения потоков применяется более точная идентификация, чтобы сократить число ложных срабатываний. По умолчанию таблица содержит 4096 ячеек (bucket). Значение можно изменить через sysctl net.core.flow_limit_table_len. Это значение применяется лишь при создании новой таблицы и смена значения не меняет активные таблицы.

Рекомендуемая конфигурация

Ограничения для потоков полезны в системах с большим числом одновременных соединений, где наличие соединения, занимающего CPU на 50%, указывает проблему. В таких средах следует включать функцию ограничения для потоков на всех CPU, обрабатывающих приёмные (rx) прерывания (как установлено в /proc/irq/CPU#/smp_affinity).

Работа функции зависит от того, насколько размер очереди входных пакетов превышает порог ограничения для потоков (50%) + размер истории потока (256). В экспериментах хорошо зарекомендовала себя установка для net.core.netdev_max_backlog значения 1000 или 10000.

RFS

RPS распределяет пакеты на основе хэш-значений, что обеспечивает хорошее распределение нагрузки, но не учитывает местоположение приложений. Эта задача решается с помощью распределения (направления) потоков (Receive Flow Steering или RFS). Целью RFS является повышение скорости обращений к кэшу данных путём направления обработки пакетов в ядре на CPU, где запущен поток (thread) приложения, являющегося получателем пакета. В RFS применяются те же механизмы, что и в RPS, для постановки пакетов в очередь невыполненных заданий другого CPU и активизации этого CPU.

В RFS пакеты не пересылаются напрямую по хэш-значению, но оно применяется в качестве индекса таблицы потоков. Эта таблица сопоставляет потоки с CPU, где они обрабатываются. Хеш потока (см. RPS) служит для вычисления индекса в таблице. В каждой записи таблицы указывается CPU, который последним обрабатывал этот поток. Если в записи ещё не указано пригодного CPU, пакеты, связанные с этой записью распределяются с помощью обычного механизма RPS. Один процессор (CPU) может быть указан в нескольких записях. При большом числе потоков и небольшом количестве CPU весьма вероятно, что один поток (thread) приложения обрабатывает потоки с разными хэш-значениями.

Глобальная таблица потоков rps_sock_flow_table указывает *желаемый* CPU для потоков - CPU, обрабатывающий поток в пользовательском пространстве. Каждое значение в таблице является индексом CPU, который обновляется при вызовах recvmsg и sendmsg (в частности, inet_recvmsg(), inet_sendmsg() и tcp_splice_read()).

Когда планировщик переносит поток (thread) на другой CPU при наличии принятых пакетов у прежнего CPU, порядок доставки пакетов может нарушаться. Для предотвращения этого в RFS применяется вторая таблица потоков для отслеживания остающихся пакетов в каждом потоке. Таблица rps_dev_flow_table создаётся для каждой аппаратной очереди приёма каждого устройства. В каждой ячейке таблицы содержится индекс CPU и счётчик. Индекс CPU представляет *текущий* CPU в очередь которого помещаются пакеты этого потока для последующей обработки ядром. В идеальном случае обработка в ядре и пользовательском пространстве выполняется одним CPU, поэтому индекс CPU в обеих таблицах совпадает. Если планировщик недавно перенёс поток (thread) пользовательского пространства, а ядро всё ещё имеет в очереди пакеты для обработки на прежнем CPU, индексы будут различаться.

Счётчик в rps_dev_flow_table указывает длину необработанной очереди текущего CPU в момент последней постановки в очередь пакета из этого потока. В каждой очереди невыполненных заданий имеется головной счётчик, инкрементируемый при извлечении пакета из очереди. Хвостовой счётчик вычисляется как сумма значения головного счётчика и размера очереди. Иными словами, счётчик в rps_dev_flow[i] указывает последний пакет потока i, который был помещён в очередь текущего назначенного CPU для потока i (значение i выбирается по хэшу и несколько потоков могут иметь одинаковое значение i).

Для предотвращения нарушений порядка пакетов при выборе CPU для обработки пакета (функция get_rps_cpu()) сравнивается таблица rps_sock_flow с таблицей rps_dev_flow очереди, в которую помещён пакет. Если желаемый для потока CPU (из rps_sock_flow) является текущим CPU (из rps_dev_flow), пакет помещается в очередь невыполненных заданий этого CPU. Когда процессоры различаются, в качестве текущего CPU устанавливается желаемый CPU, если выполняется одно из приведённых ниже условий.

- Значение головного счётчика очереди текущего CPU не меньше записанного значения хвостового счётчика в rps_dev_flow[i].
- Текущий CPU не установлен (>= nr_cpu_ids).
- Текущий CPU отключён (offline).

После этих проверок пакет передаётся текущему (возможно, обновлённому) CPU. Эти правила нацелены на то, чтобы поток переносился на другой CPU лишь в том случае, когда у прежнего не остаётся необработанных пакетов, поскольку остающиеся пакеты могли быть приняты позже тех, которые будут обрабатываться новым CPU.

Конфигурация RFS

RFS поддерживается лишь в ядрах с включённой опцией CONFIG_RPS (включена по умолчанию для SMP), однако не будет применяться без явного включения в конфигурации. Число записей в глобальной таблице потоков задаётся в файле /proc/sys/net/core/rps_sock_flow_entries, в таблицах по потокам - в файлах /sys/class/net/<dev>/queues/rx-<n>/rps_flow_cnt

Рекомендуемая конфигурация

Число записей в обоих типах таблиц должно быть задано до включения RFS для приёмной очереди с округлением вверх до ближайшей степени 2. Предлагаемое число потоков зависит от ожидаемого в любой момент числа активных соединений, которое может быть значительно меньше числа открытых соединений. Опыт показывает, что rps_sock_flow_entries = 32768 достаточно хорошо подходит для серверов со средней загрузкой.

Для устройства с одной очередью значение rps_flow_cnt для очереди обычно устанавливается равным rps_sock_flow_entries. Для устройств с несколькими очередями значение rps_flow_cnt для каждой очереди может задаваться как rps_sock_flow_entries/N, где N - число очередей. Например, при rps_sock_flow_entries = 32768 и 16 очередями приёма для rps_flow_cnt каждой очереди можно задать значение 2048.

Accelerated RFS

Accelerated RFS по отношению к RFS - то же, что RSS для RPS - механизм распределения нагрузки с аппаратным ускорением, использующий программное состояние для распределения потоков в зависимости от местоположения (CPU) потока (thread) приложения, потребляющего пакеты каждого потока. Ускоренному RFS следует работать лучше RFS, поскольку пакеты передаются напрямую CPU, локальному для потребляющего данные потока (thread). Целевым CPU будет процессор (CPU), на котором выполняется приложение, или, в крайнем случае, CPU, локальный по отношению в CPU потока (thread) приложения с иерархии кэширования.

Для включения ускоренного RFS сетевой стек вызывает функцию драйвера ndo_rx_flow_steer, чтобы сообщить о желаемой аппаратной очереди для пакетов, соответствующих определённому потоку. Сетевой стек автоматически вызывает эту функцию при каждом обновлении записи для потока в rps_dev_flow_table. Драйвер, в свою очередь, использует зависящий от устройства метод для программирования NIC в части распределения пакетов.

Аппаратная очередь для потока выводится из CPU, записанного в rps_dev_flow_table. Стек обращается к сопоставлению CPU с аппаратными очередями, которое поддерживает драйвер NIC. Это автоматически создаваемое обратное отображение таблицы близости IRQ в файле /proc/interrupts. Драйверы могут использовать функции библиотеки ядра cpu_rmap (обратное отображение близости CPU) для заполнения этого сопоставления. Для каждого CPU в сопоставлении устанавливается очередь, ближайшая к обрабатывающему CPU по местоположению кэша.

Конфигурация Accelerated RFS

Accelerated RFS поддерживается лишь ядрами, собранными с опцией CONFIG_RFS_ACCEL, а поддержка механизма обеспечивается NIC и драйвером. Требуется также включить фильтрацию ntuple с помощью ethtool. Сопоставление CPU с очередями автоматически выводится из близости IRQ, заданной драйвером для каждой приёмной очереди, поэтому дополнительной настройки конфигурации не требуется.

Рекомендуемая конфигурация

Этот механизм следует включать, когда нужно использовать RFS и NIC поддерживает аппаратное ускорение.

XPS

Распределение (направление) передачи пакетов (Transmit Packet Steering или XPS) - это механизм интеллектуального выбора очереди передачи для использования при отправке пакетов через устройство с несколькими очередями. Это можно обеспечить путём записи двух типов сопоставлений - отображения CPU на аппаратные очереди или отображения приёмных очередей на аппаратные очереди передачи.

1. XPS с использованием отображения CPU.

   Целью этого сопоставления обычно является явное назначение очередей исключительно подмножеству CPU с обработкой завершения передачи на CPU из этого подмножества. Этот подход обеспечивает два преимущества. Во-первых, значительно снижается конкуренция при блокировке очередей на устройстве, поскольку на одну очередь претендует меньшее число CPU (конкуренцию можно устранить полностью, если у каждого CPU будет своя очередь передачи). Во-вторых, снижается частота пропусков в кэше при завершении передачи, в частности для строк кэша данных, содержащих структуры sk_buff.

2. XPS с использованием сопоставления очередей.

   Это отображение применяется для выбора очереди передачи на основе конфигурации карты приёмных очередей, заданной администратором. Набор приёмных очередей можно сопоставить с набором очередей передачи (многие со многими), хотя обычно применяется сопоставление 1:1. Это позволяет передавать пакеты в одну ассоциацию очередей для приёма и передачи. Такой подход полезен при интенсивном опросе многопотоковой (multi-threaded) рабочей нагрузки, где возникают проблемы с привязкой конкретного CPU к конкретному потоку (thread) приложения. Потоки (thread) приложений не привязываются к CPU и каждый поток (thread) обслуживает пакеты, полученные в одной очереди. Номер приёмной очереди хэшируется в сокете соединения. В этой модели отправка пакетов в очередь передачи, соответствующую связанной очереди приёма позволяет снизить издержки CPU. Работа по завершению передачи фиксируется в той же ассоциации очередей, которую опрашивает данное приложение. Это избавляет от издержек, связанных с запуском прерывания на другом CPU. Когда приложение очищает пакеты в процессе опроса, завершение передачи может обрабатываться вместе с опросом в контексте того же потока (thread), что снижает задержку.

XPS настраивается по очередям передачи путём задания битовой карты сопоставления с CPU или приёмными очередями, которые могут использовать эту очередь для передачи. Обратное отображение CPU или приёмных очередей на очереди передачи рассчитывается и поддерживается каждым сетевым устройством. При передаче первого пакета из потока для выбора очереди вызывается функция get_xps_queue(), использующая идентификатор

очереди передачи сокета соединения для поиска соответствия в таблице сопоставления приёмных очередей с очередями передачи. Как вариант, эта функция может использовать идентификатор применяемого CPU в качестве ключа для поиска в таблице сопоставления CPU с очередями. Если идентификатор соответствует одной очереди, она применяется для передачи. При соответствии нескольких очередей из них выбирается одна с использованием хэш-значения потока для расчёта индекса в наборе очередей. При выборе очереди передачи на основе сопоставления с приёмными очередями очередь передачи не проверяется на соответствие приёмному устройству, поскольку это требует дорогостоящего поиска в пути данных.

Выбранная для передачи определённого потока очередь сохраняется в соответствующей структуре сокета для потока (например, соединения TCP). Эта очередь применяется для последующих пакетов, передаваемых в этот поток, чтобы предотвратить нарушение порядка пакетов (out of order или ooo). Такой подход также сокращает издержки, связанные с вызовами get_xps_queues() для пакетов потока. Для предотвращения переупорядочения пакетов очередь для последующих пакетов потока можно поменять лишь при установке для пакета в потоке skb->ooo_okay. Этот флаг указывает, что в потоке нет оставшихся пакетов и очередь передачи можно сменить без риска нарушения порядка. За установку флага ooo_okay отвечает транспортный уровень. Например, TCP устанавливает этот флаг, когда все данные для соединения подтверждены.

Конфигурация XPS

XPS поддерживается только ядрами, собранными с опцией CONFIG_XPS (задана по умолчанию для SMP). При наличии опции в ядре работа XPS зависит от включения и конфигурации XPS при инициализации устройства. Сопоставление CPU с очередями передачи можно посмотреть и изменить через файл /sys/class/net/<dev>/queues/tx-<n>/xps_cpus, сопоставление приёмных очередей с очередями передачи - через файл /sys/class/net/<dev>/queues/tx-<n>/xps_rxqs, где <dev> указывает имя сетевого интерфейса, а <n> - номер очереди передачи.

Рекомендуемая конфигурация

Для сетевых устройств с одной очередью нет смысла настраивать XPS поскольку здесь нет выбора очередей. В системах с несколькими очередями предпочтительно настраивать XPS так, чтобы каждое ядро (CPU) сопоставлялось с одной очередью. При совпадении числа CPU и очередей каждая очередь будет сопоставляться с одним CPU и конкуренции за процессорные ядра не будет. Если очередей меньше, чем CPU, лучше выбирать ядра (CPU) для данной очереди, использующие тот же кэш, который применяется CPU, обрабатывающим завершение передачи для очереди (прерывания при передаче).

При выборе очереди передачи по приёмной очереди нужно явно настроить для XPS сопоставление приёмных очередей с очередями передачи. Если сопоставление приёмных очередей с передающими не задано, очередь передачи выбирается на основе сопоставления с CPU.

Ограничение скорости по очередям передачи (TX)

Для аппаратного ограничения скорости передачи применяется атрибут max-rate, указывающий максимальную скорость в Мбит/с и задаваемый в файле /sys/class/net/<dev>/queues/tx-<n>/tx_maxrate, где <dev> указывает имя сетевого интерфейса, а <n> - номер очереди передачи. Нулевое значение, принятое по умолчанию, отключает ограничение.

Приложение

Близость SMP IRQ

Файлы /proc/irq/<IRQ#>/smp_affinity и /proc/irq/<IRQ#>/smp_affinity_list указывают CPU, разрешённые для данного IRQ#. Битовая маска (smp_affinity) и список процессоров (smp_affinity_list) указывают разрешённые CPU. Отключать все CPU не разрешается, а если контроллер IRQ не поддерживает IRQ значение будет неизменным и разрешает все CPU.

Файл /proc/irq/default_smp_affinity задаёт принятую по умолчанию маску близости, применяемую для всех неактивных IRQ. После выделения (активации) IRQ для его битовой маски близости устанавливается принятое по умолчанию значение, которое можно изменить через указанные выше файлы. По умолчанию применяется маска 0xffffffff.

Ниже приведён пример ограничения IRQ44 (eth0) процессорами CPU0-3, а затем процессорами CPU4-7 (на системе SMP с 8 ядрами).

```
# cat /proc/irq/44/smp_affinity
ffffffff

# echo 0f > /proc/irq/44/smp_affinity
# cat /proc/irq/44/smp_affinity
0000000f
```

Затем на другом хосте используется утилита ping для генерации пакетов по адресу хоста, где вносились изменения

```
# ping -f 172.16.77.3
PING 172.16.77.3 (172.16.77.3): 56 data bytes
...
--- 172.16.77.3 ping statistics ---
6029 packets transmitted, 6027 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.4 ms
```

Посмотрим статистику использования ядер для прерывания IRQ44

```
# cat /proc/interrupts | grep 'CPU\|44:'
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
44:       1068       1785       1785       1783          0          0          0          0    IO-APIC-
level  eth1
```

Как можно видеть из приведённой выше статистики прерывание IRQ44 передавалось только первым 4 ядрам (0-3). Перенаправим затем IRQ на CPU(4-7).

```
# echo f0 > /proc/irq/44/smp_affinity
# cat /proc/irq/44/smp_affinity
000000f0
```

Снова создадим на другой машине поток пакетов для хоста, где вносились изменения

```
# ping -f 172.16.77.3
PING 172.16.77.3 (172.16.77.3): 56 data bytes
..
--- 172.16.77.3 ping statistics ---
2779 packets transmitted, 2777 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.5/585.4 ms
```

Статистика использования ядер для IRQ44 будет иметь вид

```
# cat /proc/interrupts | grep 'CPU\|44:'
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
44:        1068       1785       1785       1783       1784       1069       1070       1069   IO-APIC-
level  eth1
```

Из приведённого вывода можно видеть, что значения счётчиков доставки IRQ44 изменились лишь для 4 последних ядер, а счётчики для CPU0-3 не изменились.

Литература

[1] Tom Herbert, Willem de Bruijn, Scaling in the Linux Networking Stack (https://www.kernel.org/doc/html/latest/networking/scaling.html)

[2] Ingo Molnar, Max Krasnyansky, SMP IRQ affinity (https://www.kernel.org/doc/html/latest/core-api/irq/irq-affinity.html)

[3] Hugo Krawczyk, New Hash Functions for Message Authentication (https://link.springer.com/content/pdf/10.1007/3-540-49264-x_24.pdf)

Николай Малых

nmalykh@protokols.ru