Краткое содержание

Оглавление

Вступление

Как программисту и веб-разработчику, мне нравится узнавать о последних дополнениях к различным ядрам и исследованиях в области вычислительной техники. Когда я впервые попробовала поработать с Berkeley Packet Filter (BPF) и Express Data Path (XDP) в Linux, я в них влюбилась. Это очень хорошие инструменты, и я рада, что данная книга привлекает к ним внимание, чтобы все больше разработчиков использовали их в своих проектах.

Позвольте подробнее рассказать о моей работе и о том, почему я так полюбила эти интерфейсы ядра. В свое время я выступала в качестве основного сопровождающего Docker вместе с Дэвидом. Docker, если вы не в курсе, посылает `iptables` бóльшую часть логики фильтрации и маршрутизации для контейнеров. Первым делом я выпустила для Docker патч, решающий такую проблему: `iptables` не предоставляла в CentOS одинаковые флаги командной строки, поэтому запись в `iptables` не удавалась. Было еще много загадочных проблем, похожих на озвученную, — опытные разработчики меня поймут. Кроме того, `iptables` не предназначена для соблюдения тысяч правил на хосте, к тому же это ухудшает производительность.

Затем я узнала о BPF и XDP. Познакомившись с ними поближе, я забыла о проблемах с `iptables`! Сообщество разработчиков ядра даже работает над заменой `iptables` на BPF (oreil.ly/cuqTy). Аллилуйя! Cilium (cilium.io) — инструмент для создания контейнерных сетей — также использует BPF и XDP для внутренних компонентов своего проекта.

Но это не все! BPF может сделать гораздо больше, чем просто выполнить сценарий `iptables`. С BPF вы можете отследить любую функцию `syscall` или

ядра, а также любую программу пользовательского пространства. `bpftrace` (github.com/iovisor/bpftrace) предоставляет пользователям DTrace-подобные возможности в Linux из их командной строки. Вы можете отследить все открываемые файлы и процесс, вызывающий уже открытые, подсчитать системные вызовы, выполненные программой, отследить OOM killer и многое другое... BPF и XDP используются также в Cloudflare (oreil.ly/OZdmj) и балансировщике нагрузки Facebook (oreil.ly/wrM5-) для предотвращения распределенных атак типа «отказ в обслуживании». Не буду сейчас говорить о том, почему XDP так хорош в отбрасывании пакетов, потому что вы узнаете об этом в главах, посвященных XDP и сетевым технологиям.

Я познакомилась с Лоренцо в сообществе Kubernetes. Созданный им инструмент `kubectl-trace` (oreil.ly/Ot7kq) позволяет пользователям легко запускать собственные программы трассировки в кластерах Kubernetes.

Мой любимый сценарий использования BPF — написание пользовательских трассировщиков, демонстрирующих людям, что производительность их программного обеспечения невысока или программа выполняет слишком много обращений к системным вызовам. Никогда не стоит недооценивать возможность доказать чью-то неправоту на основании объективных данных. Не волнуйтесь: эта книга поможет вам написать первую программу отслеживания. Существовавшие ранее инструменты использовали очереди потерь для отправки наборов образцов (sample set) в пространство пользователя для агрегации, а BPF — более удобный инструмент, поскольку позволяет создавать гистограммы и применять фильтры непосредственно в источнике событий. И в этом его прелесть.

По долгу службы я работаю над инструментами для разработчиков. Лучшие инструменты обеспечивают автономность интерфейсов, что позволяет реализовывать любые возможности. Процитирую Ричарда Фейнмана: «Я очень рано понял разницу между знанием названия предмета и знанием предмета»[1]. До сих пор вы могли знать только название BPF и то, что он может быть полезен для вас.

Что мне нравится в этой книге — она дает знания, которые пригодятся вам для создания новых инструментов с помощью BPF. Прочитав книгу и вы-

[1] Оригинальная цитата: I learned very early the difference between knowing the name of something and knowing something.

полнив упражнения, вы сможете использовать BPF как свою секретную суперсилу. Можете сохранить BPF в своем инструментарии на тот случай, когда он окажется наиболее необходим и максимально полезен. Вы не просто изучите BPF, но и поймете, как он работает. Прочитайте эту книгу, и вы узнаете, на что способен BPF.

Эта развивающаяся экосистема очень интересна! Я надеюсь, что она еще вырастет, так как все больше людей начинают пользоваться BPF. Я с радостью прочитаю о том, что в итоге создадут читатели этой книги, будь то сценарий для отслеживания глупой ошибки в программном обеспечении, настраиваемый брандмауэр или инфракрасное декодирование (lwn.net/Articles/759188). Обязательно сообщайте мне обо всем, что вы разрабатываете!

Джесси Фразелль

Предисловие

В 2015 году Дэвид был разработчиком ядра в Docker — компании, которая популяризировала контейнеры. Его повседневная деятельность заключалась в помощи сообществу и развитии проекта. В его обязанности входило также рассмотрение срочных запросов, отправленных членами сообщества. А еще он должен был убедиться в том, что Docker работает для всех видов сценариев, включая высокопроизводительные рабочие нагрузки, которые поступают с тысяч контейнеров, в любой момент времени.

Для диагностики проблем с производительностью в Docker мы использовали *флейм-графы*, которые представляют собой инструмент для расширенной визуализации, облегчающей навигацию по этим данным. Язык программирования Go позволяет легко измерять производительность приложений и извлекать сведения о ней с помощью встроенной конечной точки HTTP и создавать графики на их основе. Дэвид написал статью о возможностях профилировщика Go и о том, как можно использовать выдаваемые им показатели для создания флейм-графов. Существенный недостаток того, как Docker собирает данные о производительности, состоит в том, что профилировщик по умолчанию отключен. Поэтому, если вы пытаетесь отладить проблему, связанную с производительностью, первое, что необходимо предпринять, — перезапустить Docker. Основная проблема этой стратегии в том, что при перезапуске службы вы, вероятно, потеряете соответствующие собираемые данные, а затем вам придется ждать, пока вновь не произойдет

событие, которое вы пытаетесь отследить. В своей статье о флейм-графах Дэвид упомянул это как необходимый шаг для измерения производительности Docker, но не обязательно все должно происходить именно так. Эта реализация заставила его начать исследовать различные технологии для сбора и анализа показателей производительности любого приложения, в результате чего и произошло его знакомство с BPF.

Параллельно, но без участия Дэвида, Лоренцо искал возможность лучше разобраться во внутренних компонентах ядра Linux. Он обнаружил, что можно легко изучить многие подсистемы ядра, обратившись к ним с помощью BPF. Пару лет спустя, работая в компании InfluxData, он смог применить BPF для того, чтобы ускорить прием данных в InfluxCloud. Сейчас Лоренцо состоит в сообществах BPF и IOVisor и трудится в компании Sysdig над Falco.

За последние несколько лет мы использовали BPF в нескольких сценариях — от сбора данных о применении кластеров Kubernetes до управления политиками сетевого трафика. Мы узнали о его плюсах и минусах, поработав с ним и прочитав множество постов в блогах таких лидеров технологии, как Брендан Грегг и Алексей Старовойтов, и таких компаний, как Cilium и Facebook. Эти публикации очень помогли нам в свое время, а также послужили превосходным справочным материалом для создания данной книги.

Изучив множество ресурсов, мы поняли, что каждый раз, когда нужно было что-то узнать о BPF, нам приходилось перечитывать множество постов в блогах, страницы руководств и прочих материалов в Интернете. Написание этой книги — попытка собрать все знания, разбросанные в Сети, под одной обложкой, чтобы рассказать об этой фантастической технологии следующему поколению энтузиастов BPF.

Мы разделили нашу работу на девять глав, в которых демонстрируем, чего вы можете достичь, используя BPF. Можно читать отдельные главы как справочное руководство, но если вы новичок в BPF, рекомендуем знакомиться с ними по порядку. Так вы поймете основные концепции BPF и узнаете, каков его потенциал.

Мы надеемся: даже если вы эксперт в области анализа наблюдаемости и производительности или исследуете новые возможности производственных систем, то тоже почерпнете что-то для себя из этой книги.

Условные обозначения

В этой книге используются следующие условные обозначения.

Курсив

Применяется для обозначения новых понятий и терминов, которые авторы хотят особо выделить.

Шрифт без засечек

Используется для обозначения адресов электронной почты и URL-адресов.

`Моноширинный`

Используется для текста (листингов) программ, а также внутри абзацев для выделения элементов программ: имен переменных или функций, названий баз данных, типов данных, имен переменных среды, инструкций и ключевых слов, имен файлов и каталогов.

Этот значок означает совет или предложение.

Этот значок указывает на примечание общего характера.

Этот значок обозначает предупреждение.

Использование примеров кода

Сопутствующий материал (примеры кода, упражнения и т. д.) вы можете найти по адресу github.com/bpftools/linux-observability-with-bpf.

Эта книга призвана помочь вам в работе. Примеры кода из нее вы можете использовать в своих программах и документации. Если объем кода не-

существенный, связываться с нами для получения разрешения не нужно. Например, для написания программы, использующей несколько фрагментов кода из этой книги, разрешения не требуется. А вот для продажи или распространения компакт-диска с примерами из книг издательства O'Reilly нужно получить разрешение. Ответы на вопросы с использованием цитат из этой книги и примеров кода разрешения не требуют. Но для включения объемных примеров кода из этой книги в документацию по вашему программному продукту разрешение понадобится.

Мы приветствуем указание ссылки на источник, но не делаем это обязательным требованием. Такая ссылка обычно включает название книги, имя автора, название издательства и ISBN. Например: «BPF для мониторинга Linux, Дэвид Калавера, Лоренцо Фонтана (Питер), 2020. 978-5-4461-1624-9».

Если вам покажется, что использование кода примеров выходит за рамки оговоренных выше условий и разрешений, свяжитесь с нами по адресу permissions@oreilly.com.

Благодарности

Написать книгу оказалось сложнее, чем мы думали, но это было одно из самых полезных занятий в нашей жизни. Книга создавалась на протяжении многих дней и ночей, и это было бы невозможно без помощи наших партнеров, семей, друзей и домашних питомцев. Мы хотели бы поблагодарить Дебору Пейс, подругу Лоренцо, и его сына Рикардо за терпение, которое они проявили, когда он бесконечно сидел за компьютером. Спасибо другу Лоренцо Леонардо Ди Донато за советы и написание статей о XDP и тестировании.

Мы бесконечно благодарны Робин Минс, жене Дэвида, за то, что она вычитала первые наброски, с которых началась эта книга, и черновики нескольких глав, помогла ему написать много статей за те годы, что он был занят сочинительством, и смеялась над выдуманными им английскими словами.

Мы оба хотим поблагодарить всех тех, кто разработал eBPF и BPF. Особая благодарность Дэвиду Миллеру и Алексею Старовойтову за их постоянный вклад в улучшение ядра Linux и в конечном счете eBPF, а также в организацию сообщества. Спасибо Брендану Греггу за желание поделиться, энтузиазм и работу над инструментами, которые делают eBPF доступным для всех.

Команде IOVisor — за то, что они делают, и за тот вклад, который они внесли в создание `bpftrace`, `gobpf`, `kubectl-trace` и BCC. Дэниелу Боркману — за его огромную работу, в частности, над `libbpf` и инфраструктурой инструментов. Джесси Фразелль — за то, что она написала вступление и вдохновила нас и тысячи других разработчиков. Жерому Петаццони — за то, что он был лучшим научным редактором, которого мы только могли пожелать: его вопросы заставили нас переосмыслить многие фрагменты этой книги и пересмотреть свой выбор примеров кода.

Благодарим всех причастных к разработке ядра Linux, особенно тех, кто активно участвует в сопровождении BPF, за ответы на вопросы, исправления и новшества. Наконец, спасибо всем, кто участвовал в издании этой книги, включая редакторов Джона Девинса и Мелиссу Поттер, создателей обложки и корректоров, которые сделали эту книгу более читабельной.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1 Введение

За последние несколько десятилетий вычислительные системы стали не проще — наоборот, сложнее. Рассуждения о том, как программное обеспечение само себя обеспечивает, привели к созданию нескольких бизнес-категорий, и все они пытаются решить проблемы, связанные с необходимостью понять сложные системы. Один из подходов — анализ журналов данных, генерируемых приложениями, работающими в вычислительной системе. Журналы являются отличным источником информации. Они предоставляют точные данные о том, как себя ведет приложение. Тем не менее они ограничивают вас, потому что вы получаете только ту информацию, на сбор которой инженеры, создавшие приложение, запрограммировали эти журналы. Сбор любой дополнительной информации в формате журнала из любой системы может быть такой же сложной задачей, как декомпиляция программы и наблюдение за ходом выполнения. Другой популярный подход заключается в использовании метрик, позволяющих понять, почему программа ведет себя именно так и как она это делает. Метрики отличаются от журналов форматом данных: журналы предоставляют явные данные, а метрики объединяют данные, чтобы выяснить, как программа ведет себя в определенный момент времени.

Наблюдаемость — это новая практика, которая позволяет подойти к данной проблеме с другой стороны. Мы определяем наблюдаемость как способность задавать произвольные вопросы и получать сложные ответы от любой системы. Основное различие между наблюдаемостью, ведением журналов и группированием метрик — это данные, которые вы собираете. Учитывая то, что, практикуя наблюдаемость, вы должны отвечать на любой произвольный

вопрос в любой момент, единственный способ рассуждать о данных — собрать все данные, которые ваша система может сгенерировать, и скомпоновать их только тогда, когда необходимо ответить на ваши вопросы.

Нассим Николас Талеб, автор такого бестселлера, как *Antifragile: Things That Gain From Disorder*[1], популяризировал термин *«черный лебедь»*, назвав так неожиданные события, имеющие серьезные последствия, которых можно было ожидать, если бы их наблюдали до того, как они случились. В своей книге «Черный лебедь»[2] он объясняет, как наличие соответствующих данных может помочь снизить риск возникновения этих редких событий. При разработке программного обеспечения «черные лебеди» встречаются чаще, чем мы думаем, и они неизбежны. Поскольку можно предположить, что предотвратить такого рода события нельзя, то получить как можно больше информации о них — единственная возможность решить проблему так, чтобы она не оказала критического воздействия на бизнес-системы. Наблюдаемость помогает создавать надежные системы и предотвращать будущие «черные лебеди», поскольку она основана на той предпосылке, что вы собираете любые данные, которые помогут ответить на любой заданный в дальнейшем вопрос. Изучение «черного лебедя» и практика наблюдаемости сходятся в центральной точке — данных, которые вы собираете из своих систем.

Контейнеры Linux — это абстракция в дополнение к набору функций ядра Linux для изоляции компьютерных процессов и управления ими. Ядро, традиционно отвечающее за управление ресурсами, обеспечивает также изоляцию задач и безопасность. В Linux основными функциями, на которых основаны контейнеры, являются пространства имен и контрольные группы. Пространства имен — это компоненты, которые изолируют задачи друг от друга. В некотором смысле, когда вы находитесь внутри пространства имен, вам кажется, что операционная система не выполняет никаких других задач на компьютере. Контрольные группы — это компоненты, которые обеспечивают управление ресурсами. С точки зрения эксплуатации они дают вам детальный контроль над любым использованием ресурсов, таких как ЦП, дисковый ввод-вывод, сеть и т. д. В последнее десятилетие с ростом популярности контейнеров Linux

---

1 *Taleb N. N.* Antifragile: Things That Gain from Disorder. — N. Y.: Random House, 2012.

2 *Талеб Н. Н.* Черный лебедь. Под знаком предсказуемости. 2-е изд., доп. — М.: КоЛибри, 2020.

произошли изменения в том, как разработчики программного обеспечения проектируют большие распределенные системы и вычислительные платформы. Многоклиентские вычисления полностью зависят от этих функций в ядре.

Мы настолько полагаемся на низкоуровневые возможности ядра Linux, что у нас появляются новые источники информации, которые необходимо учитывать при разработке наблюдаемых систем. Ядро представляет собой систему, в которой вся работа описывается и реализуется на основе событий. Открытие файла — это своего рода событие, выполнение некоторой инструкции процессором — событие, получение сетевого пакета — событие и т. д. Berkeley Packet Filter (BPF) — это подсистема ядра, которая может проверять новые источники информации. BPF позволяет писать программы, которые выполняются безопасно, когда ядро реагирует на какое-либо событие. BPF обеспечивает безопасность, чтобы предотвратить системные сбои и вредоносное поведение каких-либо программ. BPF предоставляет новое поколение инструментов, которые помогают разработчикам систем наблюдать за новыми платформами и работать с ними.

В этой книге мы продемонстрируем возможности BPF, позволяющие сделать любую вычислительную систему более наблюдаемой. А также покажем, как писать программы BPF с использованием нескольких языков программирования. Мы поместили код для ваших программ в GitHub, поэтому не нужно копировать и вставлять его — вы найдете его в Git-репозитории к этой книге (github.com/bpftools/linux-observability-with-bpf).

Но прежде, чем начать разбирать технические аспекты BPF, посмотрим, как все начиналось.

История BPF

В 1992 году Стивен Маккейн и Ван Якобсон опубликовали статью *The BSD Packet Filter: A New Architecture for User-Level Packet Capture* («Пакетный фильтр BSD: новая архитектура для захвата пакетов на уровне пользователя»). В ней они описали способ реализации фильтра сетевых пакетов для ядра Unix, который работал в 20 раз быстрее, чем все остальные, имеющиеся на то время в области фильтрации пакетов. Пакетные фильтры имеют конкретную цель: предоставлять приложениям, которые отслеживают сетевую активность, прямую информацию из ядра. Обладая этой информацией, при-

ложения могут решить, что делать с пакетами. BPF представил два серьезных нововведения в области фильтрации пакетов:

- новую виртуальную машину (ВМ), предназначенную для эффективной работы с ЦП на основе регистров;
- возможность использования буферов для каждого приложения, способных фильтровать пакеты без копирования всей информации о них. Это минимизировало количество данных BPF, необходимых для принятия решений.

Такие радикальные улучшения заставили все системы Unix принять BPF в качестве технологии выбора для фильтрации сетевых пакетов, отказавшись от прежних реализаций, которые потребляли больше памяти и были менее производительными. Хотя они все еще присутствуют во многих производных ядра Unix, включая ядро Linux.

В начале 2014 года Алексей Старовойтов разработал расширенную реализацию BPF. Новый подход был оптимизирован для современного оборудования, благодаря чему его результирующий набор команд работает быстрее, чем машинный код, сгенерированный старым интерпретатором BPF. Расширенная версия также увеличила число регистров в виртуальной машине BPF с двух 32-битных регистров до десяти 64-битных. Увеличение количества регистров и их глубины позволило писать более сложные программы, поскольку разработчики могли свободно обмениваться дополнительной информацией, используя параметры функций. Эти изменения наряду с прочими улучшениями привели к тому, что расширенная версия BPF стала в четыре раза быстрее оригинальной реализации BPF.

Первоначальная цель создания новой реализации состояла в том, чтобы оптимизировать внутренний набор команд BPF, обрабатывающих сетевые фильтры. На этом этапе BPF все еще был ограничен пространством ядра, и только несколько программ в пользовательском пространстве могли задавать фильтры BPF для обработки ядром, такие как Tcpdump и Seccomp, о которых мы поговорим в следующих главах. Сегодня эти программы все еще генерируют байт-код для старого интерпретатора BPF, но ядро преобразует данные инструкции в значительно улучшенное внутреннее представление.

В июне 2014 года расширенная версия BPF стала доступной для пользователей. Это был переломный момент для будущего BPF. Как написал Алексей в патче с изменениями, «новый набор патчей демонстрирует потенциал eBPF».

BPF стал подсистемой ядра верхнего уровня и перестал ограничиваться только сетевым стеком. BPF-программы стали больше походить на модули ядра с сильным акцентом на безопасности и стабильности. В отличие от обычных модулей ядра, BPF-программы не требуют его перекомпиляции и гарантированно завершаются без сбоев.

Верификатор BPF, о котором мы поговорим в следующей главе, добавил необходимые гарантии безопасности. Здесь подразумевается, что любая BPF-программа завершится без сбоев и программы не будут пытаться получить доступ к памяти вне области их деятельности. Но эти преимущества сопровождаются определенными ограничениями: программы имеют максимально допустимый размер, и циклы должны быть ограничены, чтобы гарантировать, что память системы никогда не будет исчерпана неправильно написанной программой BPF.

Одновременно с изменениями, сделавшими BPF доступным из пространства пользователя, разработчики ядра добавили новый системный вызов (`syscall`) — `bpf`. Он станет центральным элементом связи между пользовательским пространством и ядром. Мы обсудим, как применять этот системный вызов для работы с программами и картами BPF, в главах 2 и 3.

Карты BPF станут основным механизмом обмена данными между ядром и пользовательским пространством. В главе 2 говорится о том, как применять эти специализированные структуры для сбора информации из ядра, а также отправки информации в программы BPF, которые уже работают в нем.

В книге в первую очередь рассматривается расширенная версия BPF. За пять лет, прошедших с момента появления расширенной версии, BPF получил значительное развитие, и мы подробно обсудим эволюцию программ BPF, карт BPF и подсистем ядра.

Архитектура

Архитектура BPF в ядре впечатляет. Далее мы еще рассмотрим конкретные детали, а сейчас, в этой главе, хотим дать краткий обзор того, как все работает.

Как мы говорили ранее, BPF — это высокоразвитая виртуальная машина, выполняющая инструкции кода в изолированной среде. В определенном смысле