# Глава 4

# ИНСТРУМЕНТЫ НАБЛЮДЕНИЯ

Операционные системы традиционно предлагали множество инструментов для наблюдения за работой системного ПО и аппаратных компонентов. Широкий спектр доступных инструментов и метрик заставляет новичков думать, что все или, по крайней мере, все важное можно наблюдать. На самом деле пробелов очень много, и анализ производительности систем превратился в искусство интерпретации и выводов: решения о необходимых действиях принимаются на основе косвенных инструментов и статистик. Например, сетевые пакеты можно исследовать по отдельности, а дисковый ввод/вывод — нет (по крайней мере, выделить отдельные блоки очень нелегко).

Благодаря появлению инструментов динамической трассировки, включая BCC и bpftrace на основе BPF, наблюдаемость в Linux значительно улучшилась. Темные углы теперь неплохо подсвечиваются, в том числе появилась возможность наблюдать отдельные операции дискового ввода/вывода с помощью biosnoop(8). Но многие компании и коммерческие продукты мониторинга еще не внедрили трассировку системы и упускают из виду открывающиеся возможности. Я был первым, кто разработал, опубликовал и объяснил новые инструменты трассировки, которые уже используются в том числе Netflix и Facebook.

Цели этой главы:

- определить инструменты статического анализа производительности и кризисных ситуаций;
- познакомить с типами инструментов и их оверхедом: подсчет, профилирование и трассировка;
- перечислить источники информации для наблюдения, включая: /proc, /sys, точки трассировки, kprobes, uprobes, USDT и PMC;
- показать, как настроить sar(1) для архивирования статистики.

В главе 1 я уже перечислил виды инструментов: подсчет, профилирование и трассировка, а также упомянул статические и динамические инструменты. В этой главе я подробно расскажу об инструментах наблюдения и их источниках данных, в том числе об инструменте sar(1), формирующем отчеты о деятельности системы, и кратко — об инструментах трассировки. Это даст базис для понимания наблюдаемости

в Linux. В последующих главах (6–11) я расскажу, как использовать инструменты и источники данных для решения конкретных проблем. В главах 13–15 мы подробно рассмотрим приемы трассировки.

В этой главе в качестве примера используется дистрибутив Ubuntu Linux. Большинство из описываемых инструментов доступны также в других дистрибутивах Linux. Аналогичные инструменты есть и для других ядер и операционных систем.

# 4.1. ПОКРЫТИЕ ИНСТРУМЕНТАМИ

На рис. 4.1 показана схема ОС Linux, на которую я нанес инструменты наблюдения за рабочей нагрузкой для каждого компонента[1].

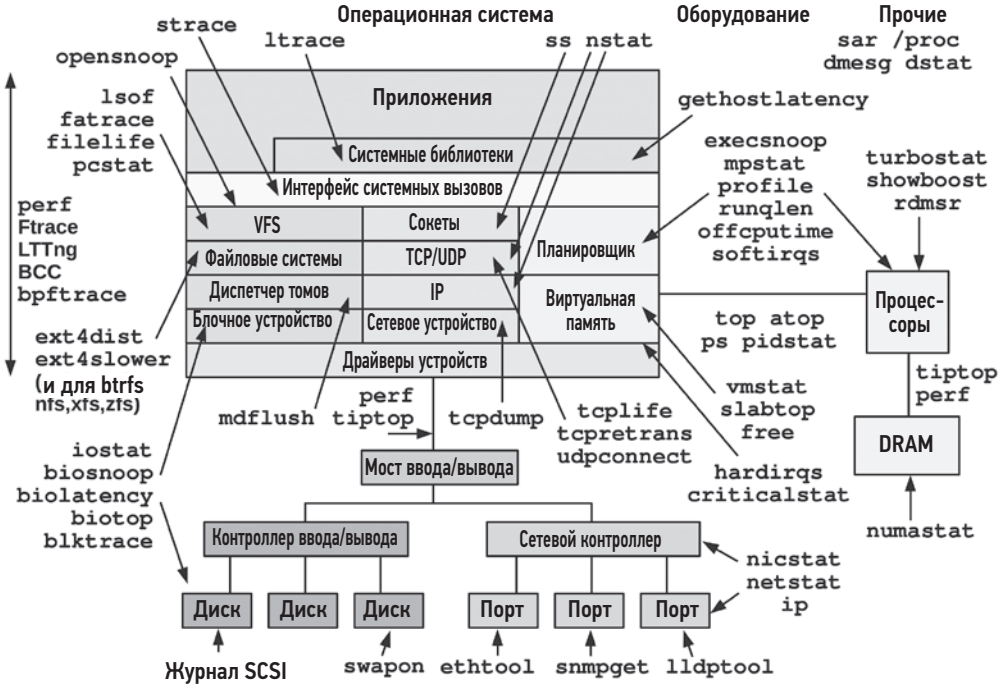


**Рис. 4.1.** Инструменты наблюдения за рабочей нагрузкой в Linux

[1] На курсах по анализу производительности, которые я вел в середине 2000-х, я рисовал свою схему ядра на доске, наносил на нее различные инструменты анализа производительности и подписывал — за чем они наблюдают. Как показала практика, такие схемы оказались эффективным средством для объяснения покрытия инструментами и формирования ментальной карты. С тех пор я не раз публиковал свои схемы в электронном виде, и теперь они украшают стены офисов по всему миру. Вы можете скачать их на моем сайте [Gregg 20a].

Большинство этих инструментов специализированы для наблюдения за конкретным ресурсом — процессором, памятью или дисками — и рассматриваются в последующих главах, посвященных отдельным ресурсам. Но есть несколько универсальных инструментов, которые могут анализировать разные области. Они будут представлены далее в этой главе: perf, Ftrace, BCC и bpftrace.

### 4.1.1. Инструменты статического анализа производительности

Есть еще один тип наблюдений, в центре внимания которого находятся атрибуты системы в состоянии покоя. В главе 2 «Методологии» в разделе 2.5.17 «Статическая настройка производительности» этот подход был описан как методология *статической настройки производительности*, а соответствующие инструменты показаны на рис. 4.2.

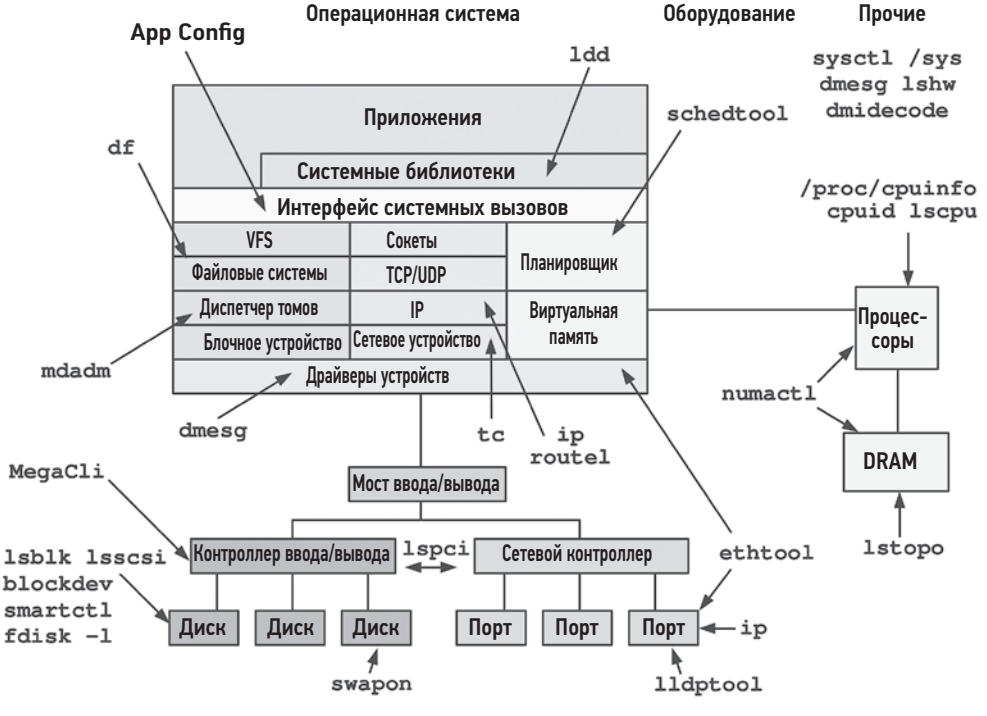


**Рис. 4.2.** Инструменты статической настройки производительности в Linux

Не забывайте использовать инструменты, перечисленные на рис. 4.2, для проверки на наличие проблем с конфигурацией и компонентами. Иногда проблемы с производительностью возникают просто из-за неправильной конфигурации.

## 4.1.2. Инструменты анализа кризисных ситуаций

Когда у вас случается критическое падение производительности, для устранения которого требуется применение различных инструментов, то может оказаться, что ни один из них не установлен в кризисной системе. Хуже того, из-за проблем с производительностью установка инструментов на сервер может занять гораздо больше времени, чем обычно длится кризис.

В табл. 4.1 перечислены рекомендуемые установочные пакеты и репозитории с исходным кодом для Linux, содержащие такие антикризисные инструменты. Имена пакетов указаны для Ubuntu/Debian, в других дистрибутивах они могут отличаться.

**Таблица 4.1.** Пакеты инструментов анализа кризисных ситуаций в Linux

| Пакет | Содержит инструменты |
|---|---|
| procps | ps(1), vmstat(8), uptime(1), top(1) |
| util-linux | dmesg(1), lsblk(1), lscpu(1) |
| sysstat | iostat(1), mpstat(1), pidstat(1), sar(1) |
| iproute2 | ip(8), ss(8), nstat(8), tc(8) |
| numactl | numastat(8) |
| linux-tools-common linux-tools-$(uname -r) | perf(1), turbostat(8) |
| bcc-tools (он же bpfcc-tools) | opensnoop(8), execsnoop(8), runqlat(8), runqlen(8), softirqs(8), hardirqs(8), ext4slower(8), ext4dist(8), biotop(8), biosnoop(8), biolatency(8), tcptop(8), tcplife(8), trace(8), argdist(8), funccount(8), stackcount(8), profile(8) и многие другие |
| bpftrace | bpftrace, basic versions of opensnoop(8), execsnoop(8), runqlat(8), runqlen(8), biosnoop(8), biolatency(8) и многие другие |
| perf-tools-unstable | версии opensnoop(8), execsnoop(8), iolatency(8), iosnoop(8), bitesize(8), funccount(8), kprobe(8) для Ftrace |
| trace-cmd | trace-cmd(1) |
| nicstat | nicstat(1) |
| ethtool | ethtool(8) |
| tiptop | tiptop(1) |
| msr-tools | rdmsr(8), wrmsr(8) |
| *github.com/brendangregg/msr-cloud-tools* | showboost(8), cpuhot(8), cputemp(8) |
| *github.com/brendangregg/pmc-cloud-tools* | pmcarch(8), cpucache(8), icache(8), tlbstat(8), resstalls(8) |

В крупных компаниях, например в Netflix, есть перформанс-команды, занимающиеся поддержкой анализа производительности ОС. Они следят за тем, чтобы все

эти пакеты были установлены в промышленных системах. По умолчанию в дистрибутиве Linux обычно устанавливаются только procps и util-linux, поэтому все остальные нужно добавить самостоятельно.

В контейнерных средах бывает желательно создать привилегированный отладочный контейнер, имеющий полный доступ к системе[1] и всем установленным инструментам. Образ этого контейнера можно установить на узлы для размещения контейнеров и разворачивать при необходимости.

Простого добавления пакетов инструментов часто бывает недостаточно: может потребоваться также настроить ядро и ПО пространства пользователя для поддержки этих инструментов. Инструменты трассировки обычно требуют включения определенных конфигурационных параметров ядра, таких как CONFIG_FTRACE и CONFIG_BPF. Для инструментов профилирования бывает необходимо, чтобы ПО было настроено для поддержки возможности обхода стека либо за счет его компиляции с поддержкой указателей фреймов (включая системные библиотеки: libc, libpthread и т. д.), либо за счет использования debuginfo-пакетов с отладочной информацией. Если ваша компания еще не сделала этого, то следует проверить работоспособность каждого инструмента анализа производительности и устранить имеющиеся недостатки до того, как эти инструменты срочно понадобятся в кризисной ситуации.

В следующих разделах мы подробно рассмотрим инструменты наблюдения за производительностью.

## 4.2. ТИПЫ ИНСТРУМЕНТОВ

Инструменты наблюдения удобно классифицировать по области покрытия — *система целиком* (system-wide) или *отдельный процесс* (per-process), а также по тому, на чем они основаны — на счетчиках или *событиях*. Пример такой классификации инструментов для Linux показан на рис. 4.3.

Некоторые инструменты присутствуют в нескольких квадрантах. Например, top(1), помимо прочего, выводит информацию о системе в целом, а инструменты анализа всей системы, основанные на событиях, часто поддерживают фильтрацию для вывода сведений о конкретном процессе (`-p PID`).

К инструментам, основанным на событиях, относятся профилировщики и трассировщики. Профилировщики наблюдают за активностью, выполняя серию моментальных снимков по событиям и создавая грубую картину цели. Трассировщики отслеживают каждое интересующее событие и могут обрабатывать их, что можно использовать, например, для создания нестандартных счетчиков. Счетчики, трассировка и профилирование были описаны в главе 1.

[1] Также можно настроить общее пространство имен для совместного использования с целевым контейнером.


**Рис. 4.3.** Типы инструментов наблюдения

В следующих разделах описываются инструменты Linux, использующие фиксированные счетчики, трассировку и профилирование, а также инструменты, выполняющие мониторинг (метрики).

## 4.2.1. Фиксированные счетчики

Ядра поддерживают различные счетчики с системными статистиками. Обычно они реализованы как целые числа без знака, которые увеличиваются при возникновении событий. Например, есть счетчики полученных сетевых пакетов, выполненных операций дискового ввода/вывода и произошедших прерываний. Они отображаются программным обеспечением для мониторинга в виде *метрик* (см. раздел 4.2.4 «Мониторинг»).

Обычно ядро поддерживает кумулятивные счетчики парами: один для подсчета количества событий, а другой для подсчета общего времени, потраченного на обработку событий. Это позволяет определить не только общее количество событий, но и среднее время (или задержку) обработки события путем деления общего времени на количество. Поскольку счетчики являются кумулятивными, то, извлекая из них информацию через определенные интервалы времени (например, каждую секунду), можно вычислить дельту, а также частоту событий в секунду и среднюю задержку. Именно так вычисляются системные статистики.

С точки зрения производительности считается, что счетчики не имеют оверхеда, потому что они включены по умолчанию и постоянно поддерживаются ядром. Единственные дополнительные расходы при их использовании — это чтение их значений из пространства пользователя (но эти расходы весьма незначительны). Далее приводятся примеры инструментов, которые читают счетчики для анализа работы системы в целом или отдельного процесса.

### *Инструменты для анализа системы в целом*

Эти инструменты исследуют активность системы в целом в контексте системного программного обеспечения или аппаратных ресурсов, используя счетчики ядра. В их число входят:

- **vmstat(8)**: статистика использования виртуальной и физической памяти в системе в целом;
- **mpstat(1)**: потребление процессора;
- **iostat(1)**: использование дискового ввода/вывода, сообщаемое интерфейсом блочного устройства;
- **nstat(8)**: статистика стека TCP/IP;
- **sar(1)**: различные статистики; их можно также архивировать для последующего сравнительного анализа.

Эти инструменты обычно доступны всем пользователям системы (не требуют полномочий root). Соответствующие статистики чаще всего отображаются в виде графиков с помощью софта для мониторинга.

Многие следуют устоявшимся соглашениям об использовании и принимают необязательные *интервал* и *количество*. Вот пример запуска vmstat(8) с интервалом в одну секунду и количеством подсчитываемых событий, равным трем:

```
$ vmstat 1 3
procs ---------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b    swpd  free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 4  0 1446428 662012 142100 5644676    1    4    28   152   33    1 29  8 63  0  0
 4  0 1446428 665988 142116 5642272    0    0     0   284 4957 4969 51  0 48  0  0
 4  0 1446428 685116 142116 5623676    0    0     0     0 4488 5507 52  0 48  0  0
```

Первая строка в выводе содержит сводную информацию — средние значения — за весь период с момента загрузки системы. Последующие строки выводятся с интервалом в одну секунду, они содержат обновленную сводную информацию и отражают текущую активность. По крайней мере такова была цель: эта версия инструмента для Linux смешивает в первой строке суммарные усредненные значения с момента загрузки и текущие (в столбцах, отражающих потребление памяти, выводятся текущие значения; подробнее о vmstat(8) рассказывается в главе 7).

### *Инструменты для анализа отдельных процессов*

Эти инструменты ориентированы на исследование отдельных процессов и используют счетчики, которые ядро поддерживает для каждого процесса. В их число входят:

- **ps(1)**: показывает состояние и различные статистики процесса, включая потребление памяти и процессора;
- **top(1)**: показывает самые активные процессы; может сортировать список процессов по потреблению процессора или другой статистике;
- **pmap(1)**: перечисляет сегменты памяти процесса со статистиками потребления.

Эти инструменты обычно читают статистики из файловой системы /proc.

## 4.2.2. Профилирование

Профилирование нужно для определения характеристик цели путем сбора образцов или моментальных снимков ее поведения. Типичной целью профилирования является потребление процессора. В этом случае профилировщик отбирает по таймеру значение указателя инструкций или трассировки стека, которые позволяют судить о путях в коде, где процесс проводит больше всего времени. Отбор образцов обычно производится с фиксированной частотой, например 100 Гц (раз в секунду), для всех процессоров в течение короткого промежутка времени, например одной минуты. Инструменты профилирования, или *профилировщики*, часто используют частоту 99 Гц вместо 100 Гц, чтобы избежать попадания в одну и ту же точку целевой активности, которое может привести к завышению или занижению оценок.

Профилирование также может основываться на не связанных по времени аппаратных событиях, таких как промахи аппаратного кэша процессора или активность шины. В этом случае профилирование помогает выявить пути, ответственные за эти события, или дает разработчикам ценную информацию для оптимизации потребления памяти их кодом.

В отличие от фиксированных счетчиков, профилирование (и трассировка) обычно производится только при необходимости, так как может вызвать довольно высокий оверхед на сбор и хранение информации. Величина таких оверхедов зависит от инструмента и частоты событий, которые он обрабатывает. Профилировщики, выполняющие сбор образцов по таймеру, обычно более безопасны: частота событий известна, поэтому оверхед можно предсказать, а кроме того, частоту событий можно выбрать такой, чтобы сделать оверхеды пренебрежимо малыми.

### *Инструменты для анализа системы в целом*

В число инструментов для профилирования системы в целом входят:

- **perf(1)**: стандартный профилировщик Linux, поддерживающий подкоманды профилирования;
- **profile(8)**: профилировщик процессора на основе BPF из репозитория BCC (описывается в главе 15 «BPF»), который подсчитывает трассировки стека в контексте ядра;
- **Intel VTune Amplifier XE**: профилирование Linux и Windows с графическим интерфейсом, поддерживает просмотр исходного кода.

Их также можно использовать для анализа отдельных процессов.

### *Инструменты для анализа отдельных процессов*

В число инструментов для профилирования отдельных процессов входят:

- **gprof(1)**: профилировщик GNU, анализирующий информацию профилирования, добавленную компиляторами (например, `gcc -pg`);

- **cachegrind**: инструмент из набора valgrind, способный профилировать аппаратный кэш (и многое другое) и визуализировать результаты с помощью kcachegrind;
- **Java Flight Recorder (JFR)**: многие языки программирования имеют свои специализированные профилировщики, которые могут исследовать контекст языка, как, например, JFR для Java.

Дополнительную информацию об инструментах профилирования ищите в главе 6 «Процессоры» и главе 13 «perf».

## 4.2.3. Трассировка

Трассировка позволяет отследить каждое событие и сохранить подробные сведения о событиях для последующего анализа или вычисления сводных данных. Трассировка похожа на профилирование, но ее цель состоит в том, чтобы собрать или исследовать все события, а не только образцы. Трассировка может иметь более высокий оверхед, чем профилирование, и существенно замедлить работу цели. Учитывайте это, принимая решение о трассировке промышленной рабочей нагрузки, а также не забывайте, что нагрузка, создаваемая трассировщиком, может исказить результаты измерения времени. Как и в случае с профилированием, трассировка обычно используется только по мере необходимости.

*Журналирование*, в ходе которого нечастые события, такие как ошибки и предупреждения, записываются в файл журнала, можно рассматривать как низкочастотную трассировку, которая включена по умолчанию. В число таких журналов входит и системный журнал.

Далее приводятся примеры инструментов трассировки системы в целом и отдельных процессов.

### *Инструменты для анализа системы в целом*

Эти инструменты трассировки исследуют активность системы в целом в контексте системного программного обеспечения или аппаратных ресурсов, используя средства трассировки ядра. В их число входят:

- **tcpdump(8)**: трассировка сетевых пакетов (использует libpcap);
- **biosnoop(8)**: трассировка блочного ввода/вывода (использует BCC или bpftrace);
- **execsnoop(8)**: трассировка событий запуска новых процессов (использует BCC или bpftrace);
- **perf(1)**: стандартный профилировщик Linux, способный также трассировать отдельные события;
- **`perf trace`**: специальная подкоманда профилировщика perf, позволяющая трассировать системные вызовы в масштабе всей системы;
- **Ftrace**: трассировщик, встроенный в Linux;
- **BCC**: библиотека трассировки и набор инструментов на основе BPF;
- **bpftrace**: трассировщик на основе BPF (bpftrace(8)) и набор инструментов.

perf(1), Ftrace, BCC и bpftrace будут представлены в разделе 4.5 «Инструменты трассировки» и подробно описаны в главах 13–15. Есть более сотни инструментов трассировки, созданных с использованием BCC и bpftrace, включая biosnoop(8) и execsnoop(8) из этого списка. Далее в книге вы найдете множество примеров их применения.

### *Инструменты для анализа отдельных процессов*

Эти инструменты трассировки ориентированы на исследование отдельных процессов, а также операционных систем, на которых они основаны. В их число входят:

- **strace(1)**: трассировка системных вызовов;
- **gdb(1)**: отладчик на уровне исходного кода.

Отладчики позволяют исследовать данные для каждого события, но для этого они должны приостанавливать и возобновлять работу цели. Это может привести к огромному оверхеду, что делает их непригодными для использования в промышленной среде.

Инструменты трассировки для анализа системы в целом, такие как perf(1) и bpftrace, поддерживают фильтры, позволяющие исследовать конкретный процесс, и могут работать с гораздо меньшим оверхедом, что делает их предпочтительным выбором, если они доступны.

## 4.2.4. Мониторинг

С понятием мониторинга мы познакомились в главе 2 «Методологии». В отличие от других типов инструментов, инструменты мониторинга ведут постоянное наблюдение и сохраняют статистики на случай, если они понадобятся позже.

### *sar(1)*

Традиционным инструментом мониторинга отдельного хоста является System Activity Reporter, sar(1), созданный на базе соответствующего инструмента из AT&T Unix. Он основан на счетчиках и имеет агент, который запускается по графику (через cron) и сохраняет состояние общесистемных счетчиков. Инструмент sar(1) позволяет просматривать данные в командной строке, например:

```
# sar
Linux 4.15.0-66-generic (bgregg) 12/21/2019        _x86_64_        (8 CPU)
12:00:01 AM     CPU     %user     %nice   %system   %iowait   %steal     %idle
12:05:01 AM     all      3.34      0.00      0.95      0.04      0.00     95.66
12:10:01 AM     all      2.93      0.00      0.87      0.04      0.00     96.16
12:15:01 AM     all      3.05      0.00      1.38      0.18      0.00     95.40
12:20:01 AM     all      3.02      0.00      0.88      0.03      0.00     96.06
[...]
Average:        all      0.00      0.00      0.00      0.00      0.00      0.00
```

По умолчанию sar(1) читает статистики из архива (если он ведется) и выводит последние сохраненные статистики. При желании можно указать временной интервал и исследовать текущую активность с указанной частотой.

sar(1) может сохранять десятки различных статистик, помогающих получить достаточно полное представление о потреблении процессора, памяти, дисков и сети, о прерываниях, энергопотреблении и многом другом. Более подробно об этих его возможностях рассказывается в разделе 4.4 «sar». Сторонние продукты мониторинга часто основаны на sar(1) или тех же статистиках, которые он использует, и позволяют получать эти метрики по сети.

### SNMP

Традиционно для мониторинга по сети используется простой протокол управления сетью (Simple Network Management Protocol, SNMP). Устройства и операционные системы могут поддерживать SNMP по умолчанию, не требуя установки сторонних агентов или инструментов экспорта. Протокол SNMP поддерживает множество основных показателей работы ОС, но не охватывает современные приложения. По этой причине в большинстве сред используются настраиваемые средства мониторинга на основе агентов.

### Агенты

Современное ПО мониторинга запускает агенты (также известные как *экспортеры*, или *плагины*) во всех системах, где требуется фиксировать показатели работы ядра и приложений. К ним относятся агенты для конкретных приложений и целей, например, для сервера базы данных MySQL, веб-сервера Apache и системы кэширования Memcached. Такие агенты могут собирать подробные метрики производительности приложений, которые нельзя получить с помощью одних только системных счетчиков.

В числе ПО мониторинга и агентов для Linux можно назвать:

- **Performance Co-Pilot (PCP)**: PCP поддерживает десятки различных агентов (называемых агентами метрик производительности: Performance Metric Domain Agents, PMDA), включая метрики на основе BPF [PCP 20].
- **Prometheus**: программное обеспечение мониторинга Prometheus поддерживает десятки различных экспортеров для баз данных, оборудования, систем обмена сообщениями, хранилищ, HTTP, API и журналов [Prometheus 20].
- **collectd**: поддерживает десятки различных плагинов.

На рис. 4.4 показан пример архитектуры мониторинга, включающей сервер базы данных хранения метрик производительности и веб-сервер для обслуживания пользовательского интерфейса клиента. Метрики передаются на сервер базы данных агентами, а затем становятся доступными в пользовательском интерфейсе клиента, где отображаются на дашбордах в виде линейных графиков или числовых значений. Примером сервера базы данных для мониторинга может служить Graphite Carbon, а примером веб-сервера, поддерживающего дашборды мониторинга, — Grafana.

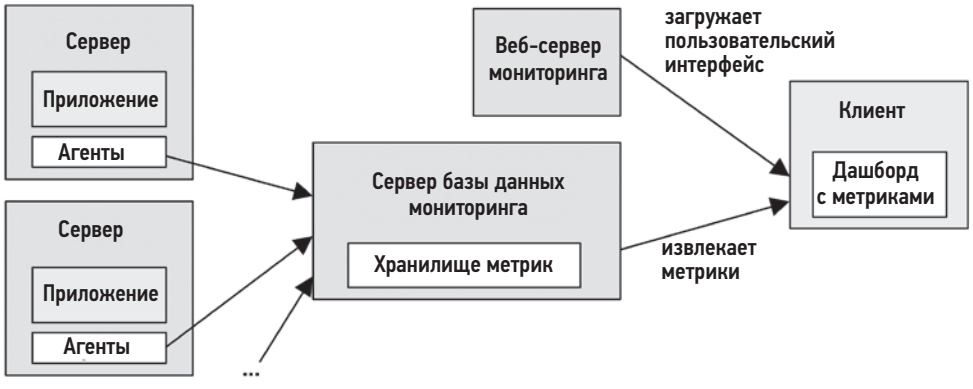


**Рис. 4.4.** Пример архитектуры мониторинга

Есть десятки продуктов для мониторинга и сотни различных агентов для разных целей. Их описание выходит за рамки этой книги. Однако у всех у них есть один общий знаменатель: системные статистики (основанные на счетчиках ядра). Системные статистики, отображаемые продуктами мониторинга, часто те же, что отображаются системными инструментами: vmstat(8), iostat(1) и т. д. Их изучение поможет понять работу продуктов мониторинга, даже если вам не приходится использовать инструменты командной строки. Эти инструменты рассматриваются в следующих главах.

Некоторые продукты для мониторинга получают метрики, запуская системные инструменты и анализируя их вывод, что неэффективно. Более совершенные продукты получают метрики напрямую, используя библиотеки и интерфейсы ядра — все то же самое, что используют инструменты командной строки. Эти источники информации рассматриваются в следующем разделе, где основное внимание уделяется наиболее типичному общему знаменателю: интерфейсам ядра.

# 4.3. ИСТОЧНИКИ ИНФОРМАЦИИ

В следующих разделах описываются различные интерфейсы в Linux, с помощью которых инструменты наблюдения могут извлекать данные о производительности. Они перечислены в табл. 4.2.

Сначала мы рассмотрим основные системные источники статистик производительности: /proc и /sys. Затем перейдем к другим источникам в Linux: учет задержек, netlink, точки трассировки, kprobes, USDT, uprobes, PMC и др.

Многие из этих источников, особенно общесистемные трассировки, используют трассировщики, описанные в главе 13 «perf», в главе 14 «Ftrace» и в главе 15 «BPF». Область, покрываемая этими источниками, показана на рис. 4.5 вместе с именами событий и групп: например, имя `block:` охватывает все точки трассировки блочного ввода/вывода, включая block:block_rq_issue.