Глава 1

МОЯ ПРОГРАММА СЛИШКОМ МЕДЛЕННАЯ

Человек входит в мой офис и говорит: «Моя программа слишком медленная». Выдержав паузу, я задаю вопрос: «А насколько медленной она должна быть?»

Хороший программист уже знает ответ на данный вопрос, поскольку описывает работу, которую необходимо проделать, и оценивает, сколько должно занимать выполнение каждой ее части. Возможно, он ответит так: «Этот запрос к базе данных обращается к 10 000 записей, из которых релевантными оказываются около 1000. Каждое обращение должно занимать около 10 мс. При этом они распределяются по 20 дискам, значит, 10 000 обращений должны в общей сложности занять около 5 с. Сетевая активность отсутствует, процессор и память заняты незначительно и простыми задачами — все это работает намного быстрее, чем обращение к диску. По факту запрос занимает около 15 с, что чересчур медленно».

Менее грамотный программист может ответить так: «Я за ночь написал 1000 строк кода, используя множество существующих библиотек, и все работает, но на запрос уходит около 15 с, а мне нужно, чтобы он занимал 1/10 с. Наверное, одна из библиотек слишком медленная. Как мне ее определить?» Если его спросить, то окажется, что он не знает, является ли 1/10 с адекватным ожиданием, сколько должен занимать каждый вызов библиотеки и правильно ли он вообще использовал библиотеки. Кроме того, также выяснится, что он не встраивал в программу способ мониторинга динамики ее кода, позволяющий определить, куда на самом деле затрачивается время. Все эти вопросы мы и будем разбирать по ходу книги.

1.1. КОНТЕКСТ ДАТА-ЦЕНТРА

Мы вводим некоторые термины и понятия из сложной программной среды. Ваша среда может быть намного проще, но все описанные идеи относятся к ней в той же степени. Терминология взята из дата-центров, но сама ее суть также применима и к базе данных, десктопным системам, транспорту, играм и прочим средам с ограничениями по времени.

Транзакция, или запрос, — это входное сообщение в компьютерную систему, которое должно быть обработано как одна единица работы. Каждый обрабатывающий транзакции компьютер называется *сервером*. *Задержка* транзакции, или *время отклика*, — это время, прошедшее между отправкой сообщения и получением результата. *Предлагаемая нагрузка* — это количество отправляемых в секунду транзакций. Когда ее показатель превышает количество транзакций, обрабатываемых за секунду, время отклика страдает, иногда значительно. *Служба* — это набор программ, обрабатывающих один конкретный вид транзакций. Большие дата-центры одновременно обрабатывают транзакции десятков разных служб, при этом каждая из них имеет свою предлагаемую нагрузку и целевое значение задержки.

Задержка транзакций не является постоянной — она имеет вероятностное распределение, взятое на основе тысячи транзакций в секунду. *Хвост распределения задержки* означает самые медленные транзакции в этом распределении. Самый простой способ обобщить время задержки — это определить задержку 99-го перцентиля: время, превышаемое 1 % самых медленных транзакций, то есть 50 транзакциями в секунду, если предлагаемая нагрузка равна 5000 транзакциям в секунду.

Под *динамикой* программы или коллекции программ мы подразумеваем активность с течением времени: какие элементы кода и когда выполняются, чего они ожидают, сколько памяти занимают и как разные программы воздействуют друг на друга. Как программисты, мы представляем себе простую динамику программы, но в реальности она может (иногда) вести себя совершенно иначе и выполняться намного медленнее, чем ожидается. Если нам удастся пронаблюдать реальную динамику, то мы сможем скорректировать наше мысленное представление и, как правило, улучшить производительность кода путем простых изменений.

Нас интересуют обращенные к пользователю транзакции в сложном ПО — к примеру, в дата-центре половины сотовых телефонов. Говоря конкретнее, нас интересуют транзакции, которые обычно выполняются быстро, но иногда занимают намного больше времени — достаточно, чтобы конечный пользователь заметил раздражающую задержку. В дата-центрах бюджет аппаратного обеспечения для каждой службы зачастую определяется тем, сколько транзакций в секунду может обработать каждый сервер. Это целевое число находится *эмпирически* путем повышения предлагаемой нагрузки до тех пор, пока не будет превышено некое установленное ограничение для хвоста распределения задержек, после чего предлагаемая нагрузка несколько снижается.

Если мы сможем понять, а затем уменьшить число излишне длительных транзакций, тогда то же самое оборудование сможет обрабатывать более объемные нагрузки в рамках целевого значения хвоста задержек без дополнительных затрат. Это может привести к серьезной экономии. Опытный инженер по эксплуатации при небольшом везении может внести простое программное изменение, которое сэкономит средства на выплату ему зарплаты в течение десяти лет. Компании и клиенты любят таких людей.

ПО, основанное на обработке ограниченных по времени транзакций, фундаментально отличается от автономных программ и программ, выполняющих пакетную обработку (или большинства бенчмарков). В транзакционном ПО важной метрикой выступает время отклика, притом что в ПО с пакетной обработкой обычно важно эффективное использование аппаратных средств. Для транзакций значение имеет не *среднее* время отклика, а самое медленное, то есть хвост распределения задержки.

> В дата-центре более высокий показатель задержек при меньшем показателе хвоста их распределения обычно является предпочтительным в сравнении с меньшей средней задержкой и более длительным хвостом. Большинство людей, приезжающих на работу из пригорода, мыслят аналогично: маршрут, который занимает на несколько минут дольше, но всегда длится одно и то же время, лучше, чем немного более быстрый маршрут, на котором иногда возникают непредсказуемые четырехчасовые задержки.

В случае ПО с пакетной обработкой средняя загруженность процессора на 98 % может считаться хорошим показателем. В случае же транзакционного ПО загрузка 98 % означает *катастрофу*, и даже 50 % загрузки процессора в среднем могут оказаться чрезмерными, поскольку это ведет к длинным откликам, когда предлагаемая нагрузка на несколько секунд резко возрастает, троекратно превышая средний показатель. Когда я только пришел работать в Google в 2004 году, средний процессор дата-центра обычно был занят на 9 % и простаивал на 91 %. Нагрузка 9 % была слишком низкой. Ее повышение до 18 % без повышения хвоста распределения задержек удвоило эффективность этих дата-центров. Очередное удваивание до 36 % оказалось бы кстати, но повышение до 72 % уже привело бы к нарушению временны́х ограничений слишком большого количества транзакций.

В этой книге, рассматривая производительность сложного ПО, основанного на транзакциях, мы предполагаем, что задействованные программы в целом работают исправно и в среднем достаточно быстро.

Мы не будем разбирать проектирование или отладку такого ПО, как и определять или улучшать его среднее быстродействие. Мы также предположим, что постоянно медленные транзакции были выявлены и исправлены в автономных тестовых/отладочных средах, не имеющих ограничений по времени, а нам остались лишь те транзакции, которые выполняются медленно иногда. Мы сосредоточимся на механизмах, которые делают такие случайные транзакции медленными, а также на том, как наблюдать эти механизмы и интерпретировать полученные наблюдения.

Когда вы используете сотовый телефон для отправки сообщения, чтения поста, поиска в Интернете, просмотра карты, трансляции видео или даже набора телефонного номера, где-то есть дата-центр, который отвечает на ваши запросы. Если эти запросы окажутся раздражающе медленными, а какое-то конкурирующее приложение или сервис будут работать быстрее, вы наверняка решите перейти на него

или по меньшей мере станете использовать медленное реже. У каждого человека в экосистеме, имеющей ограничения по времени, есть своя мотивация (зачастую финансовая) снижать досадные задержки. Однако соответствующие навыки для этого есть лишь у немногих.

Цель книги — наделить такими навыками большее количество людей.

1.2. ОБОРУДОВАНИЕ ДАТА-ЦЕНТРА

В зданиях крупных дата-центров может размещаться около 10 000 серверов, каждый из которых представляет собой компьютер размером с обычный настольный ПК, но без корпуса. Вместо этого в стойку (рэк) устанавливаются около 50 серверных плат. Всего в огромном помещении собирается 200 таких стоек. Типичный сервер содержит от одного до четырех процессорных сокетов, каждый из которых несет в себе от 4 до 50 ядер, имеет огромный объем (boatload[1]) RAM, пару жестких дисков или *твердотельных накопителей* (solid-state drives, SSD) и сетевое подключение к коммутационной сети дата-центра, чтобы все серверы могли взаимодействовать друг с другом. При этом по меньшей мере некоторые из серверов также могут взаимодействовать с Интернетом, а значит, и с вашим телефоном. Вне здания установлены большие генераторы, способные запитывать весь дата-центр, включая системы кондиционирования воздуха, в течение дней или даже недель в случае отключения электричества. Дополнительно внутри здания находятся аккумуляторы, которые могут питать серверы и сетевые коммутаторы в течение десятков секунд, пока не запустятся генераторы.

Каждый сервер выполняет множество программ. Обычно с точки зрения бизнеса нецелесообразно выделять одни серверы исключительно для обработки электронной почты, другие — для обработки карт, а третьи — только для обработки мгновенных сообщений. Вместо этого каждый сервер занят несколькими программами, которые чаще всего имеют несколько *потоков*. К примеру, серверная программа электронной почты может иметь 100 потоков воркеров, одновременно обрабатывающих почтовые запросы для нескольких тысяч пользователей, большинство из которых печатают или читают. При этом многие из активных потоков ожидают доступа к диску или выполнения работы на других слоях программы. Потоки воркеров (или просто воркеры) получают входящие запросы, выполняют запрошенные действия, отвечают и затем переходят к другому ожидающему запросу от другого пользователя. В наиболее загруженные часы практически все воркеры заняты, при этом в самый разгруженный период дня не менее половины из них простаивает в ожидании работы.

Практически во всех временны́х масштабах (микросекунда, миллисекунда, секунда и минута) происходит постоянный рост и спад предлагаемой работы. Существует

[1] Технический термин из сферы информатики, 10^{12}.

даже семидневный цикл с наименьшей активностью в субботу и воскресенье (для западной рабочей недели).

Чтобы иметь возможность контролировать время отклика, важно располагать свободными аппаратными ресурсами для обращенных к пользователю транзакций, поскольку пользовательская нагрузка имеет тенденцию резко возрастать в связи с событиями реального мира. Однако будет экономичным также иметь некоторые не обращенные к пользователю программы с пакетной обработкой, которые бы выполнялись в случае наличия свободных процессоров. Помимо обращенных к пользователю приоритетных программ и фоновых программ с пакетной обработкой, на каждом сервере всегда есть несколько управляющих программ, отслеживающих занятость сервера, количество возникающих на нем ошибок, оставшееся свободное пространство диска и т. д. Эти управляющие программы следят за состоянием серверов, а также отвечают за перезагрузку/перенастройку отдельных машин и запуск/остановку/перезапуск различных программ. По факту даже на одном сервере создается сложная среда, которая в итоге умножается на 10 000 установленных в помещении серверов.

1.3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДАТА-ЦЕНТРА

Программное обеспечение дата-центров сильно отличается от ограниченных по времени программ и бенчмарков. Оно состоит из *слоев* и *слоев подсистем*, многие из которых выполняются параллельно. Каждый слой обрабатывает запросы от множества служб и многих экземпляров отдельных служб. При этом каждый стремится отвечать на запросы достаточно быстро, чтобы вписаться в установленные ограничения времени. Зачастую слои, обслуживающие один запрос пользователя, все будут выполняться на разных серверах. Для повышения быстродействия многие слои включают программные кэши, содержащие последние обработанные данные или вычисленные результаты. Успешный поиск и повторное использование программно кэшированных данных называется *попаданием*, а неуспешный — *промахом*. Как мы увидим, динамика программного кэша может неожиданным образом влиять на задержку транзакций.

Запрос пользователя на получение текста электронного письма сначала будет перенаправлен в дата-центр, содержащий главное почтовое хранилище этого пользователя, после чего пройдет через балансирующий нагрузку сервер, который перенаправит запрос на менее занятую машину из сотни обслуживающих фронтенд электронной почты. Фронтенд-уровень ПО управляет запросом и в конечном итоге создает HTML или результат API приложения. Он запрашивает само сообщение почты с бэкенд-слоя, называемого *слоем базы данных*, который обращается к кэширующему слою БД и в случае программного промаха вызывает слой репликации (для обращения ко вторичному хранилищу почты в другом дата-центре или его обновления). В конечном итоге выполняется вызов слоя дискового сервера,

который считывает сообщение почты с одного из нескольких резервных дисков, как показано на рис. 1.1. Затем результаты возвращаются вверх по этому дереву вызовов с возможным изменением в процессе.

Рис. 1.1. Пример слоев программного обеспечения

Все это объединяется некой формой сетевой передачи сообщений, иначе называемой удаленным вызовом процедур. В этой книге мы будем использовать термин *«удаленный вызов процедур»* и его аббревиатуру RPC (от remote procedure call). RPC от одного слоя к другому могут быть синхронными, когда вызывающий ожидает ответа, или асинхронными, когда вызывающий продолжает выполнение и, вполне возможно, совершает другие RPC, которые все выполняются параллельно на разных серверах. Именно такое параллельное выполнение небольших подзадач и позволяет ПО дата-центра обрабатывать огромные объемы работы для одного запроса, успевая завершать ее за доли секунды. В обработке одной обращенной к пользователю транзакции может легко задействоваться от 200 до 2000 серверов.

На рис. 1.2 показан пример небольшого дерева RPC в стиле [Sigelman, 2010]. Сервер А может вызывать сервер Б синхронно и, получив от него ответ, вызывать В. Сервер В может параллельно вызывать серверы Г и Д, после чего ожидать от них возвращения результата.

Каждый запрос пользователя и каждый подзапрос RPC имеет ограничение по времени отклика. Если запрос пользователя на фронтенд-слое почты имеет ограничение 200 мс, то слой бэкенда может получить ограничение 160 мс, слой базы данных — еще меньшее ограничение и так далее вплоть до ограничения слоя дискового сервера 50 мс. Когда подзапрос возвращает ответ слишком медленно, каждый из вызывающих на разных слоях также вынужден отвечать медленнее. В отношении набора параллельных выполнений термин *«асимметрия выполнения»* описывает вариацию во времени завершения.

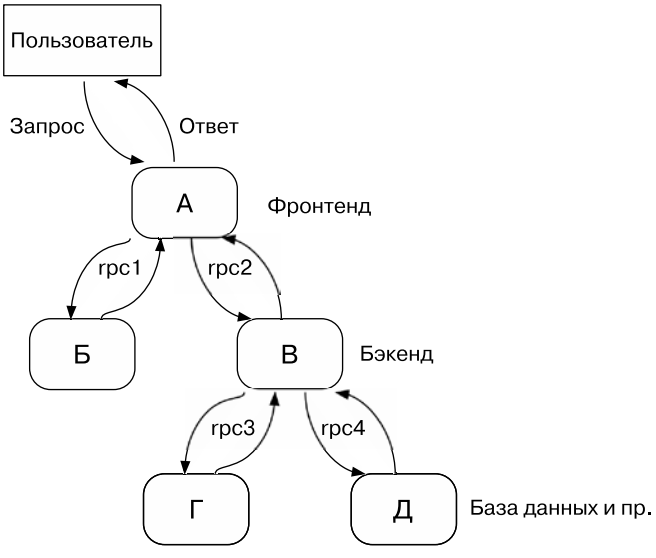

Рис. 1.2. Дерево RPC между пятью серверами А — Д

Если множество RPC выполняются параллельно, обычно самый медленный определяет общее время отклика. Таким образом, если мы выполним 100 RPC параллельно, то общее время отклика будет определяться самым медленным временем ответа 99-го перцентиля. Асимметрия выполнения делает значимым понимание и контроль продолжительных периодов отклика.

1.4. ДЛИННЫЙ ХВОСТ ЗАДЕРЖКИ

Задержка — это время, прошедшее между двумя событиями. При обсуждении задержки эти два события обязательно нужно обозначить. К примеру, «задержка RPC» может означать время между отправкой программой пользовательского режима (клиент в клиент-серверной терминологии) запроса и получением ей ответа на него. Либо оно может означать время с момента, когда вызванная программа пользовательского режима (сервер в клиент-серверной терминологии) получила запрос, до момента, когда она отправила соответствующий ответ. Эти два определения задержки, с позиции клиента и сервера, для одного RPC иногда могут иметь расхождение 30 мс и более. И за этим кроется некая загадка о том, на что затрачивается лишнее время и на какой компьютер или сетевое оборудование.

Мы по умолчанию сосредоточимся на задержке *серверных* RPC за исключением случаев, когда будем обсуждать отклонения вроде описанных выше 30 мс.

Несколько RPC к службе будут иметь разное время задержки, но для схожих запросов эти показатели зачастую будут группироваться вокруг одного значения.

Как правило, это можно обобщить в гистограмму значений задержки — небольшой график, показывающий бакеты[1] задержек на оси *X* и соответствующие этим задержкам RPC на оси *Y*.

Для транзакций дата-центра такие гистограммы задержек в нормальных случаях содержат один пик или более. В необычных же сценариях они зачастую отображают *длинный хвост* или значительно более медленное время отклика [Blake, 2015; Hoff, 2012; Weaveworks, 2017; Dean, 2013]. Гистограмма слоя дискового сервера на рис. 1.3 содержит три пика примерно на 1, 3 и 20 мс в трех видах типичных случаев. Затем в ней есть длинный хвост, уходящий за пределы 1500 мс. Желаемое время отклика составляет 50 мс или менее. Эта книга призвана помочь обрести понимание таких длинных хвостов и уменьшить их. Слабо видимые пики, возникающие незадолго до 250, 500, 750 мс и т. д., указывают на внутреннюю загадку производительности, которую мы разрешим в части II.

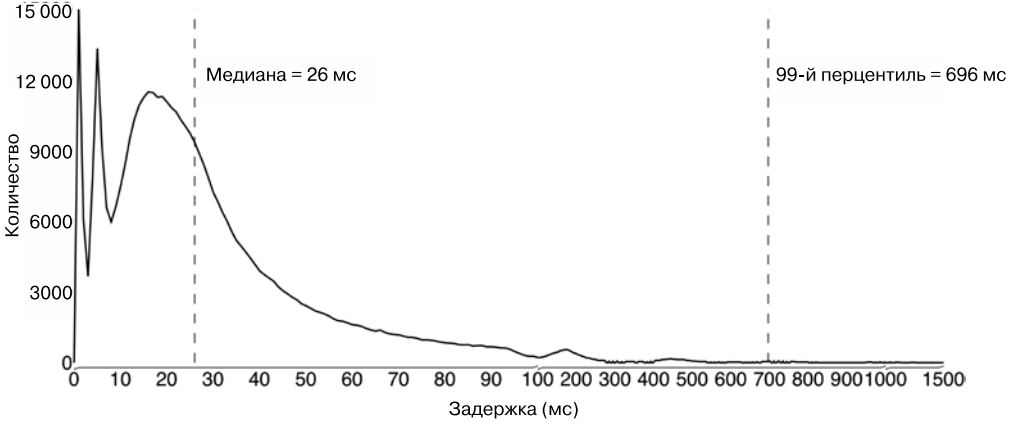

Рис. 1.3. Гистограмма обращений на уровне дискового сервера с длинным хвостом справа

В случаях, когда гистограмма задержек содержит 500 бакетов, полезно описывать ее несколькими числами вместо 500 значений. Какое же число стоит предпочесть?

Медиана (или аналогичное среднее) задержки особенно плохо подходит в качестве описания асимметричного или многовершинного распределения, поскольку редко находится около многих из фактических значений и ничего не говорит нам о форме и размере интересующего нас длинного хвоста. На графике на рис. 1.3 средняя задержка составляет 26 мс, что ничего не говорит нам о пиках или хвосте. Максимальная задержка тоже подходит плохо, так как в течение дня может иметься один крайне медленный RPC (связанный, например, с восстанавливаемой памятью или ошибкой диска), притом что все остальные будут в тысячи раз быстрее.

[1] Под бакетом (англ. bucket — «корзина») здесь подразумевается группа элементов. — *Примеч. пер.*

Вместо этого мы прибегаем к перцентилям. Если в нашей гистограмме задержек есть 50 000 измерений, то 500 самых коротких являются самым быстрым 1 %, а 500 самых длинных — самым медленным 1 %. Численная граница между самыми быстрыми 99 % и самым медленным 1 % представляет значение 99-го перцентиля — 99 % из упорядоченных измерений меньше или равны этому значению. (Существует много таких значений, все из которых лежат между 49 500-м и 49 001-м упорядоченными показателями измерений. Подойдет любое из этих чисел, но традиционно используется конкретно 49 500-е значение.) Быстрым, но при этом полезным способом описания распределения длинного хвоста будет указание значения 99-го перцентиля либо 95-го, 99,9-го и т. д. Значение 99-го перцентиля для гистограммы с рис. 1.3 составляет 696 мс, что слишком много в сравнении с целевыми 50 мс. Это представляет серьезную проблему производительности.

Из главы 9 вы узнаете, что привело к образованию конкретно этого длинного хвоста, как данная проблема была исправлена и значение итогового 99-го перцентиля, которое составило примерно 150 мс. Это простое изменение окупило мою зарплату в компании за 10 лет.

1.5. СХЕМА МЫШЛЕНИЯ

Размышляя о хвосте распределения задержек и связанных с ним проблемах производительности, мы будем следовать общепринятой в программировании практике: сначала оценивать, сколько времени должна занимать определенная работа, затем наблюдать, сколько она занимает фактически, и анализировать различия. На рис. 1.4 эта идея представлена схематично.

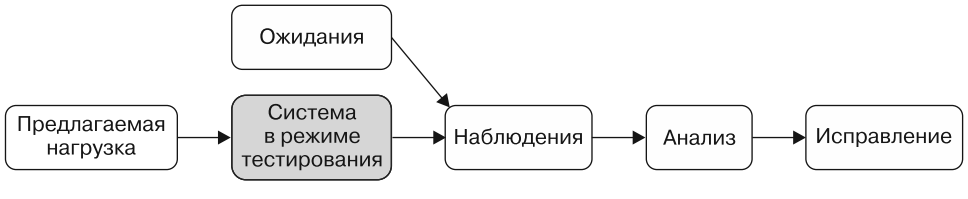

Рис. 1.4. Схема для анализа производительности сложного ПО

Данная схема состоит из тестируемой программной и аппаратной системы, некой предлагаемой нагрузки этой системы, ожиданий относительно быстродействия системы при такой нагрузке, соответствующих инструментов для наблюдения фактической динамики и производительности, анализа происходящих в системе событий и в конечном итоге исправлений или изменений, нацеленных на повышение производительности.

1.6. ОЦЕНКА ПОРЯДКА ВЕЛИЧИНЫ

При анализе производительности ПО этап выяснения «Насколько медленным оно должно быть?» подразумевает оценку того, как долго должны выполняться различные части работы. Эти оценки могут быть очень приблизительными, но все же давать полезную информацию. Программисты, занимающиеся вопросами производительности, при проектировании и написании серьезных программ постоянно оценивают в уме порядок величины.

Термин *«порядок величины»* означает приблизительное измерение размера числа. *Десятичный* порядок величины дает оценочное значение, являющееся ближайшей степенью 10 (1, 10, 100...), а *двоичный* — оценочное значение, которое является ближайшей степенью 2 (1, 2, 4, 8 и т. д.). Иногда вы будете встречать десятичные половинчатые порядки величины: 1, 3, 10, 30 и т. д. В этой книге мы используем десятичные порядки величины, если не задано иное, используя для «порядка *n*» нотацию O(*n*) и всегда указывая *единицы измерения*. Очень важно, идет ли речь о O(10) наносекундах, или O(10) миллисекундах, или O(10) байтах. Начиная с этого момента мы также будем использовать для обозначения наносекунд, микросекунд и миллисекунд сокращения *нс*, *мкс* и *мс* соответственно.

В табл. 1.1 приводятся наборы оценочных значений, с которыми должен быть знаком любой серьезный программист. Эта таблица взята из выступления Джеффа Дина (Jeff Dean), одного из немногих Google Fellows. С тех пор эти значения не слишком изменились. Я добавил столбец только порядков величины.

Таблица 1.1. Числа, которые должны знать все [Dean, 2009]

Действие	Время	O(n)
Извлечение кэша L1	0,5 нс	O(1) нс
Ошибочное прогнозирование ветви	5 нс	O(10) нс
Извлечение кэша L2	7 нс	O(10) нс
Блокировка/разблокировка мьютекса	25 нс	O(10) нс
Извлечение из основной памяти	100 нс	O(100) нс
Сжатие 1 Кбайт с помощью Zippy	3000 нс	O(1) мкс
Отправка 2 Кбайт по сети 1 Гбит/с	20 000 нс	O(10) мкс
Последовательное считывание 1 Мбайт из памяти	250 000 нс	O(100) мкс
Круговой путь внутри одного дата-центра	500 000 нс	O(1) мс
Поиск дорожки на диске	10 000 000 нс	O(10) мс
Последовательное считывание 1 Мбайт с диска	20 000 000 нс	O(10) мс
Отправка пакета CA -> Нидерланды -> CA	150 000 000 нс	O(100) мс

Выполнять оценку порядка величины ожидаемого времени выполнения различных частей программы несложно, когда у вас есть реальные измерения этого времени, позволяющие выявить, какие из них значительно отличаются от ожиданий. **Вот здесь и происходит обучение.** Иногда ваши оценки будут весьма ошибочными, в результате чего вы будете узнавать некие тонкости о работе компьютеров или программ. Иногда эти оценки будут почти верными, но программа при этом будет выполнять что-то отличающимся от вашего представления образом — такое неожиданное ускоренное или замедленное поведение требует исправления. По мере практики и совершенствования собственных навыков оценки все больше находимых вами отклонений будут оказываться реальными багами производительности.

Знание приблизительных значений из табл. 1.1 также поможет вам в определении вероятного источника отклонения быстродействия. Если выполнение некоего фрагмента программы занимает на 100 мс больше ожидаемого времени, то проблема вряд ли будет связана с ошибочным прогнозированием ветви, эффект от которого в 10 000 000 раз меньше 100 мс. Причина, скорее всего, будет относиться к времени работы диска или сети либо, как мы увидим в последующих главах, к длительным периодам удержания блокировок или ожиданию длительных подзапросов RPC.

Мы будем проектировать и создавать инструменты для наблюдения системы и отображения данных. По мере их использования вам нужно будет выработать привычку прогнозировать ожидаемые показатели в рамках порядка величины. Отточив свой навык в цикле «прогнозирование — наблюдение — сравнение», вы быстро станете замечать странные явления.

1.7. ПОЧЕМУ ТРАНЗАКЦИИ МЕДЛЕННО ВЫПОЛНЯЮТСЯ

Напомню, что нас особенно интересуют транзакции, которые обычно выполняются быстро, но иногда занимают много лишнего времени — достаточно, чтобы конечный пользователь заметил раздражающую задержку. В частности, медленные транзакции — это те, которые выходят за рамки прописанного лимита времени отклика. Что может вызывать такие задержки? То есть что может вызывать варьирующуюся задержку и в особенности длинные хвосты задержек?

Ключ здесь в том, что транзакция или определенный вид транзакций обычно выполняется быстро. Когда она оказывается медленной, это указывает на то, что, помимо нормального времени выполнения, имеется неизвестная дополнительная задержка. Если нам удастся определить ее источник, то обычно мы сможем внести в код простые изменения, которые устранят бóльшую ее часть, тем самым уменьшив хвост распределения задержек.

В многослойном программном обеспечении наиболее частым источником задержки в одном слое является то, что он ожидает ответа от другого слоя. Самый неторопливый слой может быть таковым по собственным причинам либо из-за перегрузки *неадекватно большой предлагаемой нагрузкой*. В процессе разработки лимитов для времени отклика обязательно также создавайте *лимиты на предлагаемую нагрузку*.

Изменение кода одного слоя не поможет, если он будет просто ожидать другой. Нам нужно найти самый нижний медленный слой и поработать над его улучшением. Для этого нам потребуется выполнять проектирование так, чтобы можно было наблюдать, как долго выполняется каждый слой, и преобразовывать эти измерения в отображаемые данные, которые быстро раскроют узкие места. Простая форма подобного отображения на каждом слое каждого интерфейса RPC показывает фактическую предлагаемую нагрузку относительно ее лимита, а также фактическое время отклика относительно его лимита.

Если предлагаемая нагрузка для слоя *N* приемлема и он не ожидает излишне долго нижестоящий слой *N* + 1, но его время отклика все равно слишком велико, то мы имеем RPC к слою *N*, выполняющийся на одном сервере, где задержка обычно нормальная, но временами оказывается чересчур долгой. Нам нужно понаблюдать этот конкретный сервер более пристально. Здесь медленные RPC совершают либо *лишнюю работу*, которая обычно отсутствует, либо стандартную работу, но *выполняются медленно* — медленнее, чем обычно.

Проделывание лишней работы основано на ветвистой структуре кода и состоянии, которое он поддерживает. Программы могут приходить к выполнению излишней работы совершенно по-разному, но обычно для них характерно одно общее свойство. Оно подразумевает, что эта лишняя работа выполняется, даже если программа действует на сервере автономно, без каких-либо других выполняющихся программ. Подобные баги производительности относительно легко находить путем выполнения кода офлайн в тестовой среде, снабжая его клонами или записями живых запросов и сопровождая дополнительными инструментами для поиска ошибочного паттерна ветвления. Выполнение на тестовых машинах позволяет использовать стандартные инструменты анализа производительности, которые замедляют обработку в 2 и даже в 20 раз и более. Многие инструменты наблюдения, подходящие в подобной среде, рассматриваются в книге Брендана Грегга [Gregg, 2021].

Более интересным случаем и предметом этой книги является RPC, выполняющий нормальную работу, но медленнее, чем обычно. Иными словами, нечто *вмешивается* в стандартную работу этого RPC на одном сервере, замедляя его выполнение. Такие транзакции мы называем *замедленными*. Их задержки обычно не проявляются в офлайн-тестировании, а лишь при выполнении живой, обращенной к пользователю нагрузки, зачастую только во время наиболее загруженного времени дня. Нам нужно найти источник (-и) помех и устранить его или по меньшей мере минимизировать. К сожалению, в этой живой среде инструменты наблюдения, которые замедляют обработку в два раза или даже на 10 %, слишком медленные для использования. Нам необходимы техники наблюдения и инструменты, которые в случае развертывания в дата-центрах (либо в транспортных средствах, высокозагруженных онлайн-играх и т. д.) должны задействовать не более 1 % ресурсов. В нашей индустрии таких инструментов доступно очень немного. В части III мы познакомимся с одним из них.

Напомню, что в среде дата-центра каждый сервер выполняет множество программ, каждая из которых, скорее всего, содержит множество потоков. Помехи на одном

сервере должны вызываться чем-то, работающим на нем (включая входящий и исходящий сетевой трафик). В этой среде помехи возникают почти исключительно из-за соперничества за пользование общим ресурсом.

1.8. ПЯТЬ ФУНДАМЕНТАЛЬНЫХ РЕСУРСОВ

Существует всего четыре аппаратных ресурса, совместно используемых несвязанными программами, которые выполняются на одном сервере:

- процессор;
- память;
- жесткий диск/SSD;
- сеть.

Если у программы есть множество сотрудничающих потоков, то появляется пятый ресурс:

- критическая секция ПО.

Критическая секция — это участок кода, обращающийся к общему ресурсу, который не должен одновременно использоваться более чем одним потоком. Такие участки кода защищаются программными блокировками, чтобы их мог выполнять лишь один поток одновременно. При этом любой другой поток для вхождения в эту критическую секцию вынужден ожидать завершения текущего работающего в ней потока.

Чтобы выявить помеху, вы должны понимать, как происходит нормальное выполнение. Для начала надо будет научиться тщательно измерять каждый из пяти фундаментальных ресурсов. В оставшихся главах части I описываются четыре аппаратных ресурса, а рассмотрение программных блокировок мы отложим до главы 27, когда уже разработаем необходимый инструмент наблюдения. Если медленный RPC пытается использовать любой из этих пяти ресурсов в тот момент, когда его использует другая программа или поток, то наш RPC будет вынужден подождать. Это и есть фундаментальный механизм возникновения помех или замедления.

1.9. РЕЗЮМЕ

Эта книга посвящена выработке понимания динамики выполнения программных транзакций дата-центра, базы данных, настольных систем, игр и выделенных контроллеров, в особенности тех транзакций, которые иногда занимают больше времени, чем обычно. Хороший программист способен в рамках порядка величины оценить, как долго должен выполняться создаваемый им фрагмент кода, а значит, может заметить и исправить код, который всегда работает слишком медленно. В этой книге мы предполагаем, что код, который выполняется медленно всегда, уже

исправлен. Нас интересует гораздо более сложный для понимания код, который работает медленно лишь время от времени.

На сервере дата-центра, выполняющем тысячи транзакций в секунду, некоторые транзакции будут медленными лишь иногда, совершаясь быстро при повторном выполнении. Гистограмма времени выполнения покажет нам длинный хвост медленных транзакций, которые будут непропорционально влиять на общее время отклика, наблюдаемое пользователями, и также непропорционально уменьшать количество работы, которую заданный сервер способен выполнить. Эти медленные транзакции подвержены неким внешним помехам, но в случае многослойного ПО дата-центра зачастую сложно определить, какой слой реально вносит замедление в конкретную транзакцию, а значит, и понять, где именно искать эту помеху.

Использование оценочных значений порядка величины, аналогичных показанным в табл. 1.1 (см. выше), может помочь в определении вероятных ресурсов или механизмов, вызывающих баг производительности, но обычно не позволяет обнаружить конкретный элемент медленного кода. Для этого нам потребуется спроектировать подходящие инструменты наблюдения многослойного ПО и серверов, выполняющих множество несвязанных программ, которые могут мешать друг другу.

В целом транзакция на одном сервере выполняется либо нормально, либо медленно, либо ожидает какого-то события на этом сервере. Последние два варианта обуславливаются помехой. Мы изучим механизмы этих двух сценариев, а также узнаем, как наблюдать их *на месте*.

Вот и все. Для решения проблем быстродействия, вызванных иногда медленными транзакциями, нам достаточно: 1) определить, какой слой кода работает медленно; 2) после чего выяснить, что именно ему мешает; и 3) исправить это. Оставшаяся часть книги посвящена освоению этих трех шагов. К сожалению, первые два из них будут трудными.

- Мы сосредоточимся на понимании транзакций RPC, которые выполняются медленно лишь эпизодически.
- Для 100 параллельных RPC наименьшее время 99-го перцентиля задает *общее* время отклика.
- ПО дата-центра изобилует асимметрией выполнения, что видно по длинному хвосту особенно медленных ответов.
- Замедленные транзакции говорят о вмешательстве чего-либо в работу RPC.
- Помехи возникают ввиду общего использования пяти фундаментальных ресурсов.
- Помехи сложно пронаблюдать *на месте*. Для этого мы будем создавать недостающие инструменты.
- Обязательно выполняйте оценку порядка величины ожидаемого времени выполнения, чтобы упростить обнаружение его неожиданных значений.