2 Тест производительности

Тест производительности — это программа или процесс, используемый для:

- объективной оценки производительности приложения;
- обеспечения повторяемости поведения приложения (для работы с инструментами анализа производительности).

Тест производительности выполняется до и после проведения оптимизации с целью выявить изменения в производительности. Если оптимизация не удается и производительность снижается, то программист может отказаться от неудачной оптимизации и попробовать что-либо другое. В случае повышения производительности величину этого повышения можно сравнить с ожидаемыми результатами, чтобы убедиться в успешности оптимизации. В идеальном случае производительность будет постоянно повышаться с каждой оптимизацией, но так, к сожалению, бывает не всегда. Задачей теста производительности является выявление фактов повышения и понижения производительности, чтобы можно было избежать таких модернизаций, которые затормозят выполнение приложения.

Тест производительности используется также совместно с инструментами анализа производительности. Инструменты анализа работают лучше всего тогда, когда приложение можно прогнать несколько раз совершенно одинаковым способом, тестируя таким образом одни и те же фрагменты кода. Поскольку инструменты анализа производительности обеспечивают лишь запуск и анализ приложений, только программист отвечает за то, чтобы программа выполнялась каждый раз одинаково — именно на это рассчитан тест производительности. Один или несколько тестов производительности могут служить как для измерения производительности, так и помощью инструментам анализа — все зависит от того, что лучше работает и проще в использовании.

Можно применять как готовые тесты производительности, так и написанные специально для тестирования вашего приложения. Поскольку написание собственного теста означает лишнюю работу, неплохо сначала поискать готовую программу.

Существует много стандартных тестов, таких как TPC-C[1], 3D WinBench[2] и SPEC CPU2000[3]. Эти стандартные для отрасли тесты измеряют производительность программного и аппаратного обеспечения и выдают некий показатель или набор показателей, которые могут служить для выявления изменений в производительности. Использование стандартных тестов дает дополнительное преимущество: вы можете сравнить производительность одного и того же набора приложений на различных платформах (в маркетинговых целях). Однако иногда стандартные тесты слишком обобщенные, их установка сложна или занимает неоправданно много времени, что делает более удачным выбором тест, специально созданный для конкретной ситуации. Какую бы комбинацию стандартных и специальных тестов вы ни использовали, убедитесь в том, что процесс проведения теста прост и занимает не слишком много времени (чтобы его можно было проводить часто и без проблем).

Атрибуты теста производительности

Обдумайте следующие атрибуты тестов, чтобы определить, какие из них вы будете использовать.

Воспроизводимость (обязательный атрибут)

Тест производительности, при каждом проведении выдающий разные результаты, не слишком полезен. Остановка всех других приложений, наподобие антивирусов, почтовых программ и драйверов факса, помогает получить более устойчивые результаты, однако различные переходные процессы (связанные с состоянием кэша, временными файлами, предварительно вычисленными значениями и индексами, и т. д.) могут по-прежнему приводить к тому, что приложение при последовательных прогонах будет выполняться по-разному. Более того, многие обстоятельства, неподконтрольные пользователю и приложению (такие как кэширующие контроллеры жестких дисков, фоновые задачи операционной системы и различное аппаратное обеспечение), также влияют на производительность системы и практически наверняка приведут к получению разных значений производительности.

Усреднение результатов по нескольким прогонам иногда помогает получить более устойчивые результаты измерений, но такое решение не идеально, поскольку оно может учесть то, что никак не связано с вашим приложением, например, получение входящего факса (если только ваше приложение не является драйвером факса). Более удачным вариантом может быть выбор лучшего показателя производительности по результатам нескольких прогонов (а не среднего или худшего).

---

[1] Тест TPC-C написан в Transaction Processing Performance Council. Дополнительную информацию см. по адресу http://www.tpc.org/.

[2] WinBench написан в Ziff Davis Media. Дополнительную информацию см. по адресу http://www.zdnet.com/.

[3] Дополнительную информацию см. по адресу http://www.spec.org/.

Лучший показатель явно исключает влияние вышеупомянутого факса, но скрывает такие вещи, как эффект прогрева кэша (который, возможно, тоже необходимо учитывать). Самый лучший вариант — это разобраться в переходных процессах и создать тест, проверяющий именно то, что вы хотите оптимизировать.

Репрезентативность (обязательный атрибут)

Тест производительности должен следовать типичному маршруту выполнения кода приложения, чтобы можно было анализировать и оптимизировать самые обычные ситуации. Анализ ошибочных ситуаций и нетипичных случаев приводит к обманчивым показателям производительности, последующему их включению в анализ и к бесполезным попыткам оптимизации. Лучшие тесты производительности имитируют использование приложения клиентами, поскольку только тогда вы можете быть уверены, что оптимизируете именно то, что имеет значение.

Иногда может возникнуть искушение задействовать набор тестов отдела контроля качества программ. Однако учтите, что тесты контроля качества обычно ориентированы на оценку граничных условий, ошибочных ситуаций и прочих нетипичных случаев, которые идут вразрез с тем, что обычно делают пользователи (и которые поэтому нет смысла оптимизировать). Как правило, тесты контроля качества служат для оценки функциональности программного обеспечения, а не его производительности, что делает их мало пригодными для использования в качестве тестов производительности.

Простота применения (обязательный атрибут)

Простота применения означает, что, как минимум, тест легко установить, с ним просто работать, он выполняется быстро и дает легко интерпретируемые результаты. Выполнение теста и точная интерпретация его результатов должны занимать как можно меньше времени и быть как можно проще. Чем легче будет проводить тест, тем чаще он будет выполняться разными людьми, и тем больше появится шансов на ранней стадии выявить проблемы производительности.

Проверяемость (обязательный атрибут)

Необходимо иметь возможность проверить точность теста и выдаваемых им результатов. Главным образом — точность теста. Чрезвычайно обидно потратить время на оптимизацию какой-то части приложения, а затем обнаружить, что сам тест был некорректным. Помните, что быстродействующее приложение, выдающее неверные результаты, бесполезно.

Измерение прошедшего времени (желательный атрибут)

Измерение прошедшего времени, хотя и выполняется очень часто, является не единственным возможным показателем теста производительности.

Существуют и другие показатели производительности программ. Объем расходуемой памяти, количество миллисекунд на обработку одного кадра, количество пиков в секунду или максимальное количество пользователей, которое может обслуживать система — любой из этих показателей может измеряться при тестировании производительности. Годится любой показатель, который характеризует производительность приложения. Иногда показатели, наподобие количества пиков в секунду (или другие общепринятые в отрасли показатели), для ваших целей могут оказаться лучше (чем прошедшее время), поскольку дают возможность прямого сравнения с продуктами конкурентов и использоваться в маркетинговых целях.

Полнота охвата (в зависимости от ситуации)

Тест производительности должен касаться только обычного пути выполнения кода или (что более важно) только тех путей выполнения, которые необходимо оптимизировать. Создание теста, проверяющего ошибочные условия или другой некритичный для производительности код — пустая трата времени. В некоторых случаях (драйверы и небольшие прототипы приложений) для производительности критично все приложение целиком — в этих случаях желательно полностью охватить тестом все аспекты приложения.

Точность (в зависимости от ситуации)

Оптимизированные варианты кода закрепляются в приложении только в том случае, если они приводят к удовлетворительному повышению производительности, поэтому тесты производительности должны иметь точность, позволяющую зафиксировать такое повышение. Обычно точности в один-два процента повышения производительности вполне достаточно, поскольку слишком высокая точность может привести к путанице. Сказать, что на выполнение чего-либо требуется от 18 001 119 464 до 18 784 514 894 тактов — совсем не так информативно, как сказать, что требуется 12,2 секунды.

Примеры тестов производительности

Данный раздел иллюстрирует использование тестов производительности двумя примерами.

Пример 2.1. Тест производительности сжатия

Иногда тестирование может быть таким же простым, как использование секундомера для измерения времени выполнения вашего приложения при обработке им нескольких наборов данных и запись результатов в таблицу. Не упускайте возможность сделать тест как можно проще.

Задача

Создайте тест производительности для программы HUFF.EXE, размещенной на вебсайте этой книги. В программе для сжатия файла используется кодирование Хаффмана.

Решение

В первую очередь необходимо определить, что измерять. Прошедшее время и степень сжатия кажутся очень хорошими показателями. Для того чтобы тест был репрезентативным, необходимо использовать файлы разного размера и разной степени сжимаемости. Для записи результатов теста используйте таблицу, в которой от одного прогона к другому будут отличаться только значения времени (табл. 2.1).

Таблица 2.1. Простая таблица для регистрации показателей тестирования программы HUFF.EXE

	Прогон 1	**Прогон 2**	**Прогон 3**
JPEG-файл Mars.jpg размером 510 272 байт	4,5 секунд, 484 777 байт		
Текстовый файл Constitution5.txt размером 559 140 байт	2,1 секунд, 339 969 байт		

После многократных последовательных прогонов теста значения времени оказываются очень близкими (это означает, что на результат почти не влияли какие-либо переходные процессы). На случай «неожиданного прихода факса» вы должны указать, что с каждым файлом тест должен быть прогнан по два раза, а если окажется, что значения полученных результатов заметно отличаются, то необходимо проводить дополнительные тесты вплоть до получения близких результатов.

Пример 2.2. Тест производительности сортировки

Предположим, что в ходе разработки части большого приложения ваш шеф поручил вам реализовать алгоритм, который сортирует первые *n* элементов массива случайных чисел с плавающей точкой двойной точности. Из старого учебника вы помните, что один из способов сортировки массива заключается в повторяющихся перестановках неупорядоченных элементов. Эта концепция вдохновляет вас на первую реализацию C-функции `sort1()`:

```
double a[N];
void sort1(int n) {
    int i,j;
    for (i = 0; i < n-1; i++) {
        for (j = i+1; j < n; j++) {
            if (a[i] > a[j]) {
                double tmp = a[i];
                a[i] = a[j];
```

```
                a[j] = tmp;
            }
        }
    }
}
```

Когда вы с гордостью показываете свое решение вашему коллеге, он напоминает вам, что в стандартной C-библиотеке имеется эффективный алгоритм сортировки, реализованный в виде функции `qsort()`. Эта функция может быть использована для альтернативной реализации вашей функции `sort2()`:

```
int cmp(double *d1, double *d2) {
    if (*d1 < *d2) return -1;
    if (*d1 > *d2) return +1;
    return 0;
}
void sort2(int n) {
    qsort(a, n, sizeof(double), cmp);
}
```

Для того чтобы определить, какой из этих двух алгоритмов лучше, вы создаете тест производительности. Поскольку приложение имеет дело со случайными числами, вы решаете в качестве репрезентативного набора входных данных использовать простой набор случайных чисел с плавающей точкой и не анализировать производительность сортировки для уже отсортированных или почти отсортированных массивов.

```
/* заполняем массив */
for (i = 0; i < n; i++) {
    a[i] = (double) rand();
}
```

В качестве характеристики длительности сортировки массива вы задействуете такты системных часов (см. главу 3). Просто запустите команду `rdtsc` до и после вызова алгоритма сортировки и вычислите разность двух полученных 64-разрядных значений. По определению макроса `MYSORT` будет производиться выбор метода сортировки.

```
unsigned __int64 tick1, tick2;

/* сортировка по времени */
__asm{
    rdtsc
    mov dword ptr tick1, eax
    mov dword ptr tick1+4,edx
}
```

```
#ifdef MYSORT
    sort1(n);
#else
    sort2(n);
#endif

__asm{
    rdtsc
    mov dword ptr tick2, eax
    mov dword ptr tick2+4,edx
}

printf(«length %d, ticks %I64u\n», n, tick2-tick1);
```

Важной частью теста производительности является проверка, действительно ли массив отсортирован:

```
/* результат проверки */
for (i = 0; i < n-1; i++) {
    if (a[i] > a[i+1])
        printf(«error %lf %lf\n», a[i], a[i+1]); exit(1);
}
```

Путем замеров времени выполнения обеих реализаций сортировки для массивов разной длины от 0 до 64 вы обнаруживаете, что ваше первоначальное решение имеет ту же производительность, что и метод стандартной C-библиотеки (как показано на рис. 2.1).

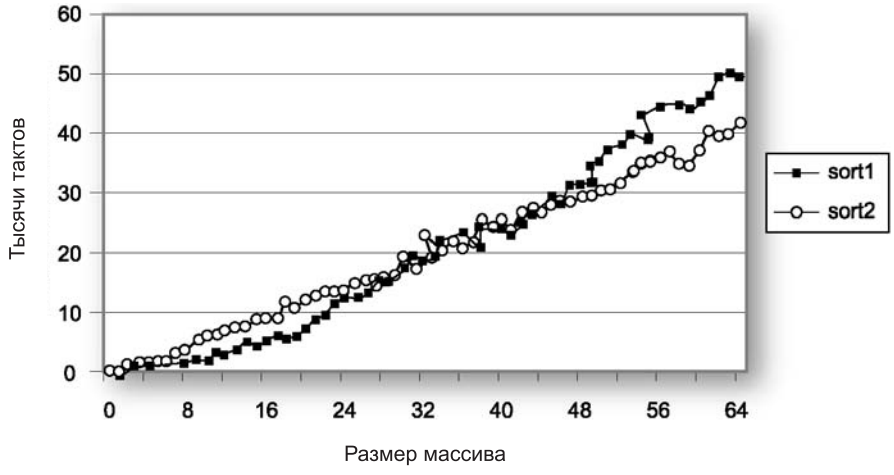

Рис. 2.1. Такты системных часов при сортировке малых массивов

Однако вас несколько беспокоят показатели при размерах массивов свыше сорока. Поэтому перед тем как помчаться к шефу со своим решением, вы проводите тесты для массивов гораздо больших размеров. Результаты показаны на рис. 2.2.

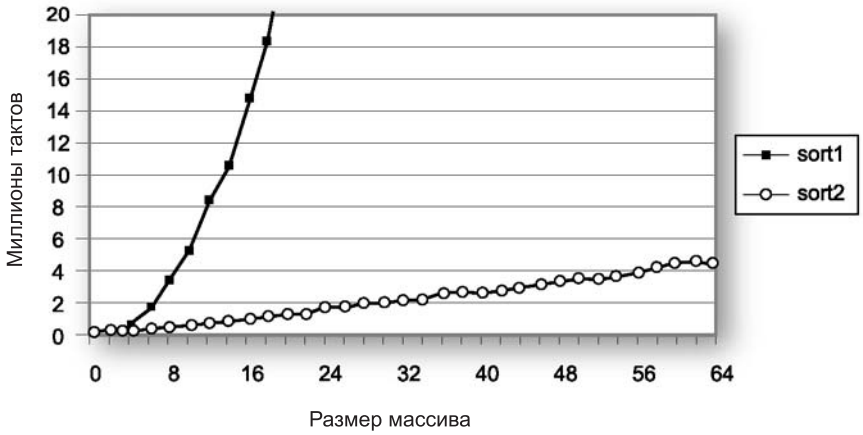

Рис. 2.2. Такты системных часов при сортировке больших массивов

Из этого графика очевидно, что алгоритм, используемый в методе `qsort()`, имеет значительно лучшие вычислительные возможности, чем ваше решение, то есть время его выполнения в зависимости от объема входных данных растет не так быстро. После консультаций с другими программистами вы выясняете, что алгоритму сортировки придется работать в основном с массивами размером около тысячи элементов. Следовательно, результаты теста настоятельно рекомендуют использовать алгоритм сортировки стандартной C-библиотеки (реализация `sort2()`) вместо вашего метода сортировки (реализация `sort1()`).

Основные моменты

Вспомните следующие основные моменты, когда начнете оптимизировать производительность при помощи теста производительности:

- Тест производительности — это программа или процесс, который используется для измерения производительности приложения и анализа производительности.
- Тест производительности должен быть, по крайней мере, воспроизводимым, репрезентативным, простым в применении и проверяемым.
- Для некоторых типов приложений уже существуют тесты производительности. Для других приложений может понадобиться написать один или более нестандартных тестов производительности.