Памятка по измерению времени при тестировании производительности алгоритмов

Самунь Виктор Сергеевич, victor.samun@gmail.com

Этап 1. Подготовка к тестированию

Для корректного измерения времени необходимо сначала подготовить окружение. Код, который подвергается анализу производительности, не должен содержать каких-либо инструкций, негативно влияющих на производительность — например, запись в лог-файл, сохранение контрольных точек для восстановления после сбоя и т.п. Вычислительный узел, на котором будут производиться замеры, также должен быть соответствующим образом подготовлен. Все процессы, способные существенно снизить скорость работы программ или внести существенную неравномерность в результаты работы (торрент-качалки, почтовые клиенты, видео- и звукопроигрыватели, чаты, автообновления системы, антивирусы и пр.) должны быть завершены.

После настройки окружения необходимо сделать несколько тестовых запусков, позволяющих оценить затраты времени на тестирование. Следует помнить, что минимальные затраты времени не гарантируют качественно выполненного тестирования, скорее наоборот — показывают, что тестирование выполнено некачественно. Размеры данных, на которых тестируется алгоритм, должны быть максимально разнообразными — как порядка нескольких килобайт, так и десятки, и сотни мегабайт (допустимы и гигабайтные данные). По выполненным тестовым запускам можно оценить границы размеров данных, и шаг, с которым размеры данных будут меняться, чтобы тестирование уложилось в разумные пределы времени.

Так как тестирование — процесс небыстрый, рекомендуется его выполнять ночью, когда вычислительный узел, в основном, простаивает или (обычно) попросту выключен и его мощности можно использовать во благо.

Этап 2. Генерация входных данных

Целью тестирования производительности алгоритмов могут быть как анализ работы одного алгоритма на разных данных, так и сравнение алгоритмов между собой. Если наша цель — сравнить между собой производительность нескольких алгоритмов, то мы обязаны сравнивать время работы на одинаковых данных. В частности, по этой причине данные лучше сгенерировать заранее и затем их использовать. Также необходимо учитывать (вне зависимости от цели тестирования), что один алгоритм на одном и том же размере данных, скорее всего, будет иметь разное время работы в зависимости от самих данных. Например, линейный поиск в массиве из n элементов может выполнять как 1 операцию, так и n операций сравнения. Алгоритм быстрой сортировки Хоара в некоторых случаях может выполнять $O(n^2)$ операций, хотя в среднем он имеет сложность $O(n \log n)$. Чтобы учесть такие особенности, можно поступить следующим

образом. Для каждого размера данных сгенерируем набор из 3 видов данных — случайные данные, лучшие и худшие. Данные являются для алгоритма худшими, если на них алгоритм работает максимально долго.

Данные d^* являются худшими данными размера n для алгоритма alg, если

$$\forall d \ \big(|d| = n \ \Rightarrow \ \text{time}\big(\text{alg}(d)\big) \le \text{time}\big(\text{alg}(d^*)\big)\big).$$

Заметим, что худшие тестовые данные позволяют сделать оценку сверху времени работы алгоритма. Понятие лучших данных определяется аналогично, такие данные позволяют сделать оценку снизу времени работы алгоритма.

Задача нахождения классов лучших и худших данных решается при помощи анализа алгоритма и логических рассуждений. Более подробно данную информацию можно прочитать в книге Кормена «Алгоритмы. Построение и анализ».

Этап 3. Подготовка к измерению времени

Предположим, что сделан один запуск алгоритма на некоторых данных и измерено время его выполнения. Вопрос в том, стоит ли считать данное время временем работы алгоритма на этих данных. Если время измеряется для выполнения примерных расчётов или оценки времени тестирования на этапе 1, то так считать, безусловно, можно, но следует в полной мере осознавать неточность данного измерения.

Алгоритм на одних и тех же данных в разных запусках будет работать разное время, т.к. имеет место погрешность измерений, которую следует оценить. Разные функции получения системного времени имеют разную точность, она обычно указана в документации языка или используемой библиотеки, из-за этого бывает невозможно замерить время выполнения короткого участка кода. Например, функции, возвращающие текущее системное время обычно имеют точность в 100мс. Для решения проблемы измерения времени, меньшего или сравнимого с минимальной точностью измерений обычно делают N запусков алгоритма на одинаковых данных и результатом будет являться $1/N$ от полученного времени выполнения. Запуски должны выполняться на одинаковых данных, время подготовки данных не должно входить во время работы алгоритма — для этого можно либо нужным образом измерять время, либо дополнительно замерить время подготовки данных и сделать поправку результата. Также желательно перед началом измерений сделать пару холостых запусков (без замеров времени), чтобы исключить влияние кешей процессора на результат измерений (запуск на прогретом кеше), которое может изменить результат измерения на порядок.

Пусть вычислено время работы алгоритма на некоторых данных указанным способом. Ясно, что если это же измерение проделать ещё несколько раз, то результаты будут получены несколько другие. Естественным образом возникают два вопроса — какое время считать результатом измерения и какова его погрешность.

Пусть сделано N запусков алгоритма и получены следующие времена работы: $(t_1, \ldots, t_N)$. Если необходимо измерить минимально возможное время работы алгоритма на указанных данных, то в качестве результата можно взять $t = \min\limits_{i=1,\ldots,N} t_i$, т.к. случайные всплески активности других программ и другие воздействия не могут улучшить время выполнения и данный результат показывает время выполнения с минимумом сторонних факторов. Обычно же измеряют не минимально возможное время работы, а среднее время работы $\left(\bar{t} = \frac{1}{N}\sum\limits_{i=1}^{N} t_i\right)$ в реальных

условиях (но с небольшим, по возможности, влиянием сторонних факторов), т.к. минимальное время получить бывает затруднительно.

Для оценки погрешности измерений вычисляют стандартное отклонение по формуле:

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(t_i - \bar{t})^2}.$$

Заметим, что для одного запуска ($N = 1$) стандартное отклонение обращается в бесконечность, т.е. результат является недостоверным и его нельзя использовать при дальнейшем анализе. Действительно, в случае ошибки измерения доверять результату нельзя. Количество запусков обычно подбирается таким образом, чтобы стандартное отклонение не превышало 5% от среднего времени работы, в противном случае результат измерений не является достоверным.

После вычисления стандартного отклонения вычисляют доверительный интервал — интервал, накрывающие неизвестный параметр, в данном случае время работы, с заданной надёжностью. Для расчёта применяют следующую формулу:

$$\Delta = t_{N-1,\alpha}\frac{S}{\sqrt{N}}.$$

Параметр α задаёт вероятность того, что неизвестный оцениваемый параметр находится в интервале $(\bar{t} - \Delta; \bar{t} + \Delta)$. Значения $t_{N-1,\alpha}$, требуемые для расчёта, приведены в таблице:

N	6	11	16	21	26	31	36	41	51	101
$t_{N-1,95}$	2.5706	2.2281	2.1314	2.0860	2.0555	2.0423	2.0301	2.0211	2.0086	1.9840

На данном этапе необходимо создать (написать простенькую библиотечку или использовать готовую) программное окружение, которое позволит по данным получить 2 числа: время работы алгоритма на этих данных и значение Δ выполненного измерения.

Этап 4. Измерение

Для выполнения измерений рекомендуется написать несколько сценариев оболочки, которые выполнят необходимые запуски и сохранят результаты работы в файл для дальнейшей обработки. На данном этапе категорически воспрещается пользоваться вычислительным узлом, чтобы не внести погрешности в измерения.

Этап 5. Обработка результатов

Важный этап тестирования — обработать полученные результаты, ибо без обработки они представляют собой бессмысленный набор чисел. Данных при тестировании обычно получается довольно много, по этой причине табличное представление результатов не является наглядным, гораздо нагляднее результаты видны на графиках.

На графике по оси абсцисс откладывается размер данных, по оси ординат — время работы. Алгоритмы между собой корректно сравнивать только в пределах одного класса данных

(лучший случай, худший и случайный данные), поэтому для каждого из этих классов разумно построить отдельный график. Для выполнения сравнения графики лучше наложить друг на друга. Грубой ошибкой является соединение точек-результатов эксперимента линиями! Во-первых, не известно, имеются ли какие-то особенности «между» полученными результатами — зависимость может быть степенной, кусочно-линейной, разрывной — какой угодно и это никому не известно. Во-вторых, если соединён линией результат эксперимента на данных размера L и $L+1$, то не очень ясен физический смысл точки для данных размера $L+0.5$, которого, очевидно, не существует.

Однако, полученные измерения можно приблизить некоторой функцией и затем сравнивать эти функции и их параметры для разных алгоритмов. Графики данных функций допустимо изображать линией, т.к. эта функция не является результатом измерения. Рассмотрим на примерах, как выполняется приближение результатов измерения некоторой функцией.

Пример 1. Линейная функция

Пусть мы сделали измерения в точках $(x_1, \ldots, x_N)$ и получили результаты $(y_1, \ldots, y_N)$. Построенный график напоминает прямую линию и мы хотим максимально хорошо приблизить эти данные линейной функцией $f(a, b; x) = ax + b$.

Словосочетание «максимально хорошо приблизить» означает, что суммарное среднеквадратичное отклонение измерений и значения функции должно быть минимальным, т.е.

$$g(a, b) = \sum_{i=1}^{N} \big(y_i - f(a, b; x_i)\big)^2 \to \min .$$

Нам осталось подобрать параметры линейной функции так, чтобы данное условие выполнялось. Необходимым условием минимума является обращение в нуль частных производных, т.е.

$$\begin{cases} \dfrac{\partial}{\partial a} \displaystyle\sum_{i=1}^{N} \big(y_i - (ax_i + b)\big)^2 = 0, \\ \dfrac{\partial}{\partial b} \displaystyle\sum_{i=1}^{N} \big(y_i - (ax_i + b)\big)^2 = 0. \end{cases}$$

Решение данной системы линейных уравнений оставим в качестве упражнения, приведём только ответ:

$$\begin{cases} a^* = \dfrac{\displaystyle\sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i - N \sum_{i=1}^{N} x_i y_i}{\Big(\displaystyle\sum_{i=1}^{N} x_i\Big)^2 - N \sum_{i=1}^{N} x_i^2}, \\ b^* = \dfrac{\displaystyle\sum_{i=1}^{N} y_i - a^* \sum_{i=1}^{N} x_i}{N}. \end{cases}$$

Педанты могут заметить, что мы не проверили достаточное условие минимума — матрица квадратичной формы дифференциала второго порядка функции $g(a, b)$ в точке (a^*, b^*) должна быть положительно определённой. Оставим проверку этого факта в качестве несложного упражнения. □

Пример 2. Степенная функция

Пусть функция f из предыдущего примера имеет вид $f(a, b; x) = ax^b$. Сведём задачу к предыдущей логарифмированием. Тогда $\widetilde{f}(a, b) = \log a + b \log x$ и к задаче применимы все рассуждения из предыдущего примера. □

Пример 3. Экспоненциальная функция

Наиболее простым способом выявления экспоненциальной зависимости может быть переход к логарифмической шкале на оси ординат и дальнейшей проверки линейности полученного графика.

На логарифмической шкале, в отличие от линейной шкалы, приращение экспоненциальное. Говоря простым языком, если отметки на линейной шкале имеют вид $(1, 2, 3, \ldots)$, то на логарифмической — $(1, 10, 100, \ldots)$. □

На самом деле, зачастую выполнять расчёты из примеров 1 и 2 вручную не нужно. Существуют специализированные средства, например, программа Origin 7.5, позволяющая выполнить аппроксимацию и вычислить точность приближения. Приём, рассмотренным в примере 3, применить проще (средства для построения графиков обычно имеют возможность использования логарифмической шкалы) и его используют для наглядного изображения экспоненциальной зависимости.

При нанесении экспериментальных результатов на график возле каждой точки необходимо указать доверительный интервал при помощи вертикального отрезка, по длине в точности равного интервалу.

Далее, после выполнения аппроксимации результатов измерений известными функциями, можно проводить сравнение характеристик алгоритмов.

Этап 6. Написание отчёта

Отчёт должен включать в себя:

1. Постановку задачи;
2. Параметры вычислительного узла, на котором выполнялось тестирование (процессор, память, ос);
3. Краткое описание тестируемых алгоритмов;
4. Результаты измерений в виде графиков или таблиц;
5. Обоснование результатов;
6. Анализ результатов;
7. В приложении — реализация алгоритмов, данные, на которых выполнялось тестирование.

Приложение. Чеклист корректности тестирования

№	Название проверки	Статус
1.1	В алгоритме нет ненужных вычислений	
1.2	На вычислительном узле завершены все некритичные для тестирования и ресурсоёмкие задачи	
1.3	Выполнен тестовый запуск, сделаны предварительные замеры времени	
1.4	Составлен план тестирования	
1.5	План тестирования содержит разнообразные размеры данных	
1.6	Оценено время тестирования, если слишком велико или слишком мало — п.1.4	
2.1а	Охарактеризован лучший случай данных для алгоритма	
2.1б	Сгенерированы лучшие данные	
2.2а	Охарактеризован худший случай данных для алгоритма	
2.2б	Сгенерированы худшие данные	
2.3	Сгенерированы случайные данные	
3.1	Подготовлено окружение для корректного, с учётом погрешности, измерения времени работы алгоритма	
3.2	Для каждого измерения выполняется несколько холостых запусков	
3.3	Написаны автоскрипты запуска тестирования	
4.1	Запущено тестирование	
5.1	В отчёте на графиках экспериментальные точки не соединены линиями	
5.2	На экспериментальные точки нанесены доверительные интервалы	
5.3	Выполнена аппроксимация экспериментальных результатов функциями	
6.1	В отчёт включена постановка задачи	
6.2	В отчёт включены параметры вычислительного узла	
6.3	В отчёт включены все результаты измерений	
6.4	В отчёте есть обоснование экспериментальных результатов	
6.5	Приложение к отчёту содержит готовую среду для воспроизведения тестов	