Eric Lippert
posted 6 years ago

C# Performance Benchmark

BEGINNER

C# Performance Benchmark Mistakes, Part Two

So far in this series we've learned about the jitter and how it compiles each method in your program "on the fly"; we'll come back to jitter issues in the next episode; in this episode I want to look at some actual (terrible) code for measuring performance.

Let's suppose part of a larger program will involve sorting a long list of integers, and we'd like to know how much time that is going to take. Rather than testing the entire "real" program and trying to isolate the specific part, we'll write a benchmark that just answers the question at hand.

There are a lot of things wrong with the code below that we'll get to in later episodes; the issue I'm going to focus on today is:

Mistake #5: Using a clock instead of a stopwatch.

```
using System;
using System.Collections.Generic;
class P
{
  public static void Main()
  {
    var list = new List<int>();
    const int size = 10000;
    var random = new Random();
    for(int i = 0; i < size; ++i)
      list.Add(random.Next());
    var startTime = DateTime.Now;
    list.Sort();
    var stopTime = DateTime.Now;
    var elapsed = stopTime - startTime;
    Console.WriteLine(elapsed);
  }
}
```

That looks pretty reasonable, right? We create some random data to test. Let's assume that ten thousand elements is about the typical problem size that the customer will see. We are careful to time only the actual sort and not include the creation of the random data. If we run it on my laptop I see that it reports about 3 milliseconds. How accurate is this?

That result is off by a factor of more than two from the correct answer; the sort actually takes about 1.2 millseconds. What explains this enormous discrepancy, and how did I know what the correct answer was?

Well, suppose you were trying to time this program by actually watching a grandfather clock that ticks once per second. This is of course ludicrous because you know what would

Likes 8 Views 13k

ABOUT THE AUTHOR

Eric Lippert
Follow

Eric Lippert designs C# analyzers at Coverity and formerly designed languages at Microsoft. Learn more at http://ericlippert.com.

necessary precision.

Subscribe

`DateTime.Now` is only slightly better than that grandfather clock: it only ticks every few milliseconds. Worse, the rate at which it ticks is not fixed from machine to machine. On modern Windows machines you can expect that the tick resolution will be around 100 ticks per second or better under normal circumstances, which is a resolution of 10 milliseconds or less. Apparently things went well for me on this laptop and I got a resolution of 3 milliseconds, which is great for `DateTime.Now` but still nowhere near fast enough to accurately measure the time spent sorting.

The correct tool to use when writing benchmarks is `Stopwatch` in the `System.Diagnostics` namespace. (I note that this namespace is well-named; everything in here is good for diagnosing problems but should probably not be used in "production" code.) Let's take a look at the code written to use `Stopwatch` instead: (There is still *plenty* wrong with this benchmark, but at least we're improving.)

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
class P
{
  public static void Main()
  {
    var list = new List<int>();
    const int size = 10000;
    var random = new Random();
    for(int i = 0; i < size; ++i)
      list.Add(random.Next());
    var stopwatch = new System.Diagnostics.Stopwatch();
    stopwatch.Start();
    list.Sort();
    stopwatch.Stop();
    Console.WriteLine(stopwatch.Elapsed);
  }
}
```

Notice first of all how much more pleasant it is to use the right tool for the job; the `Stopwatch` class is designed for exactly this problem so it has all the features you'd want: you can start, pause and stop the timer, computing the elapsed time does not require a subtraction, and so on.

Second, the results now show far more accuracy and precision: this run took 1.1826 milliseconds on my laptop. We have gone from ~10 millisecond precision to sub-microsecond precision!

How precise is the stopwatch, anyway? It depends on what kind of CPU you have; the `Stopwatch` class actually queries the CPU hardware to get such high precision. The `Stopwatch.Frequency` read-only field will tell you what the resolution is; on my laptop it is two million ticks per second, which you'll note is considerably larger than the operating system's weak promise of around 100 ticks per second for the grandfather clock that is `DateTime.Now`. This laptop is pretty low-end; better hardware can provide even higher resolution than that. Remember, one two-millionth of a second is still *hundreds* of processor cycles on modern hardware, so there's room for improvement.

Suppose for the sake of argument though that the code we were benchmarking was going to run for seconds, minutes or even hours. In those cases the difference between a resolution of 100 ticks per second and 2 million ticks per second is irrelevant, right? If you're going to be measuring minutes then you don't need the second hand to be super accurate, so why not use `DateTime.Now` in those cases?

It's still a bad idea because `Stopwatch` has been specifically designed to solve the problem of easily measuring time spent in code; `DateTime.Now` was designed to solve a completely

For example, suppose you use `DateTime.Now` to measure the time in a long-running performance benchmark that you run overnight on a machine in Seattle. Say, on Sunday November 3rd, 2013. The result you get is going to be wrong by one hour, and might in fact even be a *negative number* because `DateTime.Now` pays attention to Daylight Savings Time changes.

Just don't even go there. `DateTime.Now` is the wrong tool for the job, was designed to solve a different problem, is harder to use than `Stopwatch`, and has thousands or millions of times less precision. Avoid it entirely when writing benchmarks in C#.

Next time in this series we'll take a closer look at how the jitter can affect your benchmarks.

Read related posts in this series:

- C# Performance Benchmark Mistakes, Part One

Like this post

TECH PRO

Dear Tech Professional,

Tech.pro has been temporarily deactivated. We will bring Tech.pro back online after our team has grown in sufficient size to support the community.

Sincerely,
Tech.pro Team

Login

Email Address

Password

Login

Register

Email Address

First Name Last Name

Password

☐ I agree to Tech.Pro's Terms Register

Comments (5)

Timwi posted 6 years ago

> I note that this namespace is well-named; everything in here is good for diagnosing problems but should probably not be used in "production" code.

Then how do I run an external process in production code?

Reply

Khalid Abuhakmeh posted 6 years ago

Funny we wrote the same post basically. I would say great minds think alike, but that would be cliche..... oh hell great minds think alike. :P

Hi Eric, looking forward to the rest of the series.

Reply

Craig Wagner posted 6 years ago

Are you going to address the oft-repeated information that says that Stopwatch is inaccurate on modern multi-core CPUs?

Reply

Howard Andresier posted 6 years ago

Hi Eric. Your first code section contains an error. "stopwatch.Stop();" should probably read "var stopTime = DateTime.Now;".

I presume you wanted to demonstrate the DateTime behaviour before getting onto the Stopwatch behaviour, but you got ahead of yourself slightly. Regards, Howard.

Reply

FEED	PEOPLE	MORE
Tutorials	Profiles	How TechPro Works
Blogs	Authors	About Us
Links	Projects	Contact Us
Q&A	My Network	Site Map
Invite Others		

Blog

Twitter

Linkedin

Facebook