When studying performance, you need to watch out not only for performance degradation, but also unexpected performance improvement

devblogs.microsoft.com/oldnewthing/20121123-00

November 23, 2012

Raymond Chen

In addition to specific performance tests run by individual feature teams, Windows has a suite of automated performance tests operated by the performance team, and the results are collated across a lot of metrics. When a number is out of the ordinary, the test results are flagged for further investigation. The obvious thing that the performance metrics look for are sudden drops in performance. If an operation that used to consume 500KB of memory now consumes 750KB of memory, then you need to investigate why you're using so much memory all of a sudden. The reasons for the increase might be obvious, like "Oh, rats, there's a memory leak." Or they might be indirect, like "We changed our caching algorithm." Or "In order to address condition X, we added a call to function Y, but it turns out that function Y allocates a lot of memory." Or it could be something really hard to dig up. "We changed the timing of the scenario, so a window gets shown before it is populated with data, resulting in two render passes instead of one; the first pass is of an empty window, and then second is a when the data is present." Chasing down these elusive performance regressions can be quite time consuming, but it's part of the job. (End of exposition.) The non-obvious thing is that the performance metrics also look for sudden *improvements* in performance. Maybe the memory usage plummeted, or the throughput doubled. Generally speaking, a sudden improvement in performance has one of two sources. The first is the one you like: The explained improvement. Maybe the memory usage went down because you found a bug in your cache management policy where it hung onto stale data too long. Maybe the throughput improved because you found an optimization that let you avoid some expensive computations in a common case. Maybe you found and fixed the timing issue that was resulting in wasted render passes. (And then there are two types of explained improvements: the expected explained improvement, where something improved because you specifically targeted the improvement, and the unexpected explained improvement, where something improved as an understood side-effect of some other work.) The second is the one that you don't like: The unexplained improvement. The memory usage activity went down a lot, but you don't remember making any changes that affect your program's memory usage profile. Things got better *but you don't know why*. The danger here is that the performance gain may be the result of a bug. Maybe the scenario completed with half the I/O activity because the storage system is ignoring flush requests. Or it completed 15% faster because the cache is returning false cache hits. At the end of the day, when you finally understand what happened, you can then make an informed decision as to what to do about it. Maybe you can declare it an acceptable degradation and revise the performance baseline. ("Yes, we use more memory to render, but that's because we're using a higher-quality effects engine, and we consider the additional memory usage to be an acceptable trade-off for higher quality output.") Maybe you will look for an alternate algorithm that is less demanding on memory usage, or bypass calling function Y if it doesn't appear that condition X is in effect. Maybe you can offset the performance degradation by improving other parts of the component. Maybe the sudden performance improvement is a bug, or maybe it's an expected gain due to optimizations. But until you know *why* your performance profile changed, you won't know whether the change was good or bad.

After all, if you don't know why your performance improved, how do you know that it won't degrade just as mysteriously? Today, you're celebrating that your memory usage dropped from 200MB to 180MB. Two weeks from now, when the mysterious condition reverts itself, you'll be trying to figure out why your memory usage jumped from 180MB to 200MB.

Raymond Chen

Follow