What are the odds that two pull requests get completed at the exact same time?

devblogs.microsoft.com/oldnewthing/20180326-00

March 26, 2018

Raymond Chen

The so-called Fall Creators Update of Windows 10 encompassed four million commits in half a million pull requests. A project of this size exposes issues in infrastructure by stressing it to levels previously considered unimagineable.

One of the issues uncovered by hosting the Windows repo on Visual Studio Team Services (VSTS) is that of pull requests being completed simultaneously. As noted in the linked article, the odds of this happening are low on average-sized teams, but for a project as large as Windows, it is a persistent problem. What happens is that multiple pull requests attempt to complete, one of them successfully merges, and the others fail. The people who get the failure push the *Complete* button again, and the cycle repeats: One succeeds, and the others fail. And all during this time, still more pull requests are joining the party.

You get into a situation where there are continuous race conditions, and while it's true that forward progress is made (one of the pull requests will complete successfully), you unfortunately also create the risk of starvation (some poor guy loses the race ten times in a row), and you burn a lot of CPU cycles calculating merges that end up being thrown away.

The article explains that to address this problem, VSTS added support for queueing, so that pull request completions are serialized rather than using a lock-free algorithm.

I know that one of the ground rules of the Internet is *Don't read the comments*, but I read the comments on the Ars Technica article anyway. There was some disbelief that two pull requests could complete at exactly the same time. One will certainly be a fraction of a second sooner than the other. So what's this issue with simultaneous pull request completions?

The deal is that completing a pull request takes a nonzero amount of time. For the Windows repo, calculating the merge result can take from five seconds for a small change to significantly longer for larger changes. The index file itself is over a quarter of a gigabyte; you're spending a good amount of time just for the I/O of reading and writing that monster. And then you have to read and compare a ton of trees, and then merge the differences. All this work takes time, and if the calculations for the completions of multiple pull requests

overlap, then the one who finishes first wins, and everybody else loses. It's not random who loses the race; it's biased against people who have changes that affect a large number of files. Without queueing, somebody trying to complete a large commit will consistently lose to people with little one-line fixes.[1]

Bonus chatter: In a discussion of Windows source control, one person argued that git works great on large projects and gave an example of a large git repo: "For example, the linux kernel repository at roughly eight years old is 800–900MB in size, has about 45,000 files, and is considered to have heavy churn, with 400,000 commits."

I found that adorable. You have 45,000 files. Yeah, call me when your repo starts to get big. The Windows repo has over three million files.

Four hundred thousand commits over eight years averages to around thirteen commits per day. This is heavy churn? That's so cute.

You know what we call a day with thirteen commits? "Catastrophic network outage."

(Commenter Andrew noted that it actually averages to 130 commits per day, not 13. But 130 commits in one day would still count as a catastrophic network outage.)

[1] Note that this race applies only to pull requests targeting the same branch, but there is enough activity even within a single branch that these collisions are frequent.

Clarification: I'm not scoffing at the linux kernel. I'm scoffing at the person who told the Windows team "I don't understand why you aren't using git. Git can totally handle large projects. Why, it can even handle a project as large as the linux kernel!"

Raymond Chen

Follow