Queue everything and delight everyone

This is a blog post I've had simmering in my brainmeats for well over a year or two. I'm suddenly inspired to break blog-radio-silence and get it out of my head.

From Let the microblogs bloom - RussellBeattie.com:

> *Once this is widely accepted (and I'm sure there are many that would argue with me), the thing that will separate these types of services won't be whether they stay up (ala Twitter), but how fast your subscription messages are updated. Some services might be smaller or offer more features but not update as quickly whereas others will pride themselves on being as close to real-time as possible. The key is that it's all about messaging, not publishing. (Oh, and this also facilitates federation as well, but that's another topic).*

See also: Rearchitecting Twitter: Brought to You By the 17th Letter of the Alphabet - random($foo)

One of the problems it seems most modern web apps face is the tendency to want to do everything all at once, and all in the same code that responds directly to a user. Because, while you're in there building a user interface, it's *easy* to implement everything else that needs to happen in that same UI module or library.

Someone wants to post a bookmark? Someone wants to post a message? Well, of course you want the system to cross-reference and deliver that new piece of User Generated Content through every permutation of tag, recipient, keyword, and notification channel supported by your system.

But, do you *really* have to do everything all at once—while the person who generated that content is tapping his or her foot, waiting for the web interface to respond with feedback? Are all of these things immediately vital to the person watching the browser spin, *right now*?

No. Your user wants to get on with things. He or she wants to see the submitted content get accepted and, as feedback and confirmation, see it reflected in a personal view immediately. Does it matter—to *this person*, at *this moment*—whether it shows up *simultaneously* in a friend's inbox, the public timeline, a global tag page, or even an RSS or Atom feed?

Again, no, simultaneity doesn't really matter—because no human beings actually appreciate it. Instead, imagine a ripple effect of concentric social and attention contexts with associated people spreading out from the original submission. (This probably rates the creation of a diagram someday.)

- To make the person who's submitting something happy, offer feedback visible in their own personal context in under 50-200 milliseconds. (That is, less than half-a-second at worst, in people terms.)
- The next person to delight is someone following the first person's published content—and humanly speaking, delays of *tens of thousands of milliseconds* can be acceptable here. (That is, 1-10 seconds at worst, in people terms.)
- Finally, you can start worrying about strangers, allowing the content to propagate to tag pages, keyword tracking pages, and other public views—and I'd assert that delays of *hundreds of thousands of milliseconds* are acceptable here. (That is, 1-2 minutes at worst, in people terms.)

The idea here is that the social structure can help you scale, while still delighting people. Even with these delays, the system is still better at getting the word out than the original content creator would be at notifying all the others involved with an out-of-band system like IM or email. And that's at worst—on most good days, all the delays should tend to be on the order of seconds or less.

And how do you do all of this? Use queues. Sure, the original submission of content can and should be done all at once—just enough to get the content into the user's collection. Then, queue a job for further processing and get out of the way. In fact, just queue one job from the user interface—the processor of *that* queue can then queue further jobs for all the other individual processing tasks that are likely susceptible to plenty of parallel processing and horizontal scaling.

Meanwhile, the original user creating content has been thanked for their submission and life goes on. In fact, their life may include going on to submit many more pieces of content in rapid succession, thanks to your delightfully responsive web user interface.

And, in the end, that's really the purpose of a web-based content creation interface—accepting something as quickly as possible to make the user happy enough to continue submitting more. The other part of the user interface, retrieval, serves simply to get the original content distributed as fast as can be reasonably expected.

Now, preparing for fast retrieval is another story. The flip side to processing queues are message inboxes—expect content duplicated everywhere and fetched simply, rather than using cleverly expressed SQL joins that bring a system to its knees. But, that's another post altogether. :)

bookmark this on Delicious saved by 155 other people tags: architecture scalability queue messaging programming scaling twitter performance development

This entry was written by l.m.orchard, posted on July 4, 2008 at 3:17 pm, filed under entries and tagged delicious, laconica, queues, scaling, twitter, webdev. Bookmark the permalink. Follow any comments here with the RSS feed for this post. Post a comment or leave a trackback: Trackback URL.

« on what should I hack next?