Realistic time estimates for progress bars etc

Asked 15 years, 9 months ago Modified 1 year, 3 months ago

Viewed 5k times

16

I know I am not the only one who does not like progress bars or time estimates which give unrealistic estimates in software. Best examples are installers which jump from 0% to 90% in 10 seconds and then take an hour to complete the final 10%.

Most of the time programmers just estimate the steps to complete a task and then display *currentstep/totalsteps* as a percentage, ignoring the fact that each step might take a different time to complete. For example, if you insert rows into a database, the insertion time can increase with the number of inserted rows (easy example), or the time to copy files does not only depend on the size of the file but also on the location on the disk and how fragmented it is.

Today, I asked myself if anybody already tried to model this and maybe created a **library with a configurable robust estimator**. I know that it is difficult to give robust estimates because external factors (network connection, user runs other programs, etc) play their part.

Maybe there is also a solution that uses profiling to set up a better estimator, or one could use machine learning approaches.

Does anybody know of advanced solutions for this problem?

In connection to this, I found the article Rethinking the progress bar very interesing. It shows how progress bars can change the perception of time and how you can use those insights to create progress bars that seem to be faster.

EDIT: I can think of ways how to manually tune the time estimate, and even with a 'estimator library' I will have to fine tune the algorithm. But I think this problem could be tackled with statistical tools. Of course, the estimator would collect data during the process to create better estimates for the next steps.

What I do now is to take the average time something took in the previous step (steps grouped by type and normalized by e.g. file size, size of transaction) and take this average as estimate for the next steps (again: counting in different types and sizes).

Now, I know **there are better statistical tools** to create estimators and I wonder if anybody applied those to the problem.

language-agnostic statistics progress-bar machine-learning estimation

Share Improve this question Follow

edited Mar 27, 2009 at 14:15

asked Mar 27, 2009 at 13:47

f3lix

29.9k ● 11 ● 67 ● 86

7 Answers

Sorted by: Highest score (default)

8

While an undergrad, Julian Missig and I ran an experiment not unlike the Harrison et al. paper. As you might expect for a class project, we didn't really get enough data to make strong claims, except that for a 5-second interval, showing no progress bar was actually perceived to be *shorter*.

So, if the task is likely to take shorter than say 10 seconds, it's best not to show a progress bar at all. That's not to say that you shouldn't show *any* feedback, but a progress bar is likely to just make it seem slower.

If you're interested, Julian has the paper and poster on his site.

Share Improve this answer

answered Mar 27, 2009 at 14:48

Follow

Daniel Dickison
21.9k ● 14 ● 70 ● 89

7

Thank goodness I'm not the only one!

I don't know of a library that handles estimation, but I can personally vouch for your profiling ideas. I once implemented a progress bar which was used to report the progress of a long, complicated file operation (several small files were being read, processed, and then combined into a larger file). I had the software keep track of the time it took for reads, writes and processing, and then adjusted the progress bar accordingly. After the program had been run a couple of times, the progress bar would move as smooth as silk. No pauses and no fast blips.

This works as long as the time taken for your operations are easily measured. I would be leery of using this method on something like a download progress indicator, since the speed of the network is completely indeterminate.

Share Improve this answer

edited Sep 15, 2009 at 19:53

Follow

answered Mar 27, 2009 at 13:58

James Eichele
119k ● 41 ● 182 ● 214

I don't think the problem is that they estimate the number of steps so much as it's that often the wrong definition of "step" is used. In your example of the installer going from 0 to 9% in 10 seconds and then an hour for the rest, I've seen that happen when the programmer decided to count the number of files to copy, not the number of bytes.

Say there were 10 files, 9 of them were 5K each (readme, license, icon, etc.) and the last was a 2Gig ISO, well, the first 9 would copy really fast and the last would be slow! Counting files was the wrong thing to count, should have counted bytes. Problem is, if you want to count bytes then you need to implement your own copy routine so you can provide status updates during the copy of the large file. Is is really worth it to implement your own copy routine?

The other problem is that an install (like many other things) is made up of stacks of routines that can be quite deep. These routines can do a lot of things, but they're likely generic routines, and have nothing in them that is capable of updating some progress meter at a much higher level. You'd need to reimplement a number of common routines to get good progress information.

As for a robust estimator, I think that would be really hard. The particular steps could be defined in a config file, but you'd need to have progress updates from every part of the install process. Additionally, the time to do these things would obviously vary from machine to machine, so you'd likely be way off anyway. Of course, once you've

done the install on a specific machine you could likely estimate the install on that machine next time. ;-)

Share Improve this answer

Follow

answered Mar 27, 2009 at 14:04

Walden Leverich

4,516 ● 2 ● 23 ● 30

3

The problem with using a progress bar is often a process takes multiple different steps. So if I were doing a progress dialog for a software update, I wouldn't use a single progress bar, but a task list with check marks so the user can see what task is being performed at the moment.

Put a progress bar next to the task if it's taking longer than 10 seconds so they can see that work is being done and they don't abort it too early.

Download Update
Stop running processes
Update software
Configure software
Restart program

Individual tasks are nice because past performance strongly indicates future performance. The download's first 10 seconds will likely show you how long the remainder of the file takes. Same with the update itself.

Shorter processes don't need a progress bar, so don't even display one on any process until that one process

has taken 10 seconds or more. That way the user on a fast system just sees a checkmark on each task, and on a slow system the user sees the checkmarks, and if it stays on a task 'too long' they get the progress bar with actually useful information.

And the progress bar doesn't make any promises about how long the later tasks will take.

Having an overall "estimated time remaining" at the bottom that covers best guess for all tasks is very useful, but I wouldn't show that on a progress bar.

The thing about progress bars is they are meant to travel linearly. When they jump and stutter it's very frustrating for the user - they are actually less useful and give the wrong information in that case.

Pick the right tool for the job. Too many times a progress bar is chosen when it's actually the wrong tool.

edited Sep 16, 2023 at 21:36

halfer

20.4k ● 19 ● 108 ● 200

answered Mar 27, 2009 at 15:58

Adam Davis

93.5k ● 60 ● 271 ● 333

As you say, you may have 100 steps but each of those steps will take a different amount of time depending on what they do.

2

One approach would be to group tasks by what they're doing (deleting, changing registry values, downloading, copying files, etc) and for each group, assigning some key properties:

- Which monitorable metrics apply (copy speed, unpacking speed, etc)?
- What's an average worst-case rate for that process?

Then you need to build a list of what you're going to be doing for the whole job, eg:

1. Unpacking a 100meg file (group: unpacking, value:100)
2. Copying out 120megs (group: copy, value:120)
3. Setting registry values (group: registry, value:25)
4. Clean up (group: deletion, value:100)

So from that you can work out an overall "estimate" based on your preset average worst-case values but the key to accuracy is updating each metric multiplier as you learn how fast *that system* can do each task.

It took Microsoft a decade to get it right so don't be too distressed if it doesn't work at first =)

Share Improve this answer

Follow

answered Mar 27, 2009 at 14:00

Oli

240k ● 65 ● 226 ● 303

That's probably how I would do it too, but wouldn't it be nice to have a drop-in solution for this, or at least some sort of tool support? I just don't want to spend days on optimizing a progress bar ... :-) – f3lix Mar 27, 2009 at 14:04

If there isn't a drop-in, I'd assume it's because progress bars track so many different types of operation and even each op can vary in size (eg DBIO perf is dependant on statement complexity as well as overall DB perf). Very specific for everything but simple file operations. – Oli Mar 27, 2009 at 14:11

Well, the drop-in wouldn't work without configuration. But maybe you would be able to say eg: those steps depend on the input size, those follow a logarithmic series distribution, and those are just unpredictable... – f3lix Mar 27, 2009 at 14:20

2

Another (and much simpler way) is to just pad the estimate and the user perception.

Most progression bars are there more for UI responsiveness than duration prediction: the user needs to have feedback confirming that the program isn't stalled - but doesn't care that much about completion time.

If I'm waiting for a task, and it goes to 50% completion in 10 seconds - I'm getting frustrated when it takes another 20 seconds to complete the last 50%.

For the same task, if it goes to 50% in 30 seconds, keeps going until 60% - and then magically jumps to 100% - I'm surprised, but not annoyed.

If the task is really short or completely unpredictable, some animated loop works as well (browser loading icon, iPhone wait animation, etc...).

If you are in the couple cases where you really need accuracy - then it's probably worth spending some time in the code for better reliability of the bar.

Share Improve this answer

Follow

answered Mar 27, 2009 at 15:41

ptyx

4,164 ● 1 ● 20 ● 21

2

I'm using DREJ to do non-linear least-squares regression on historical progress. It works pretty well.

I use a database table to store my historical data. I re-build my estimator function based on the last 100 entries in the table.

I have annotations on long-running methods to identify the rate determining variable.

YMMV, but the next time the estimate takes that into account.

Share Improve this answer

Follow

edited Nov 26, 2009 at 9:13

f3lix

29.9k ● 11 ● 67 ● 86

answered Nov 25, 2009 at 23:18

Tim Williscroft

3,735 ● 25 ● 37