This is a preview of a research series titled ‘DAOsclarosis’. This first preview is published here with the remainder of articles posted on other forums. Feedback in comments or via email/DM’s is always welcomed. LaTeX isn’t enabled yet on the forums so it may be easier to read on our notion page

Posted on our Notion: Notion – The all-in-one workspace for your notes, tasks, wikis, and databases.

DAOsclarosis: on the persistence of faulty models in governance

“Demosclerosis isn’t a problem you solve It’s a problem you manage.

” Jonathan Rauch, DEMOSCLEROSIS The Silent Killer of American Government, 1994

The DAO Corollary

F.K.A. Amdahl’s Corollary

The most efficient way to implement a piece of software is to do it all yourself.

No time is wasted communicating (or arguing); everything that needs to be done is done by the same person, which increases their ability to maintain the software; and the code is by default way more consistent.

Turns out “more efficient (alt

effective

)” doesn’t mean “faster (both in performance and time to delivery)”. When there are more people working on the same problem, we can parallelized more at once.

When we break work up across a team, in order to optimise for the team, we often have to put more

work in, individually, to ensure that the work can be efficiently parallelized. This includes explaining concepts, team meetings, code review, pair programming, etc. But by putting that work in, we make the work more parallelized, speeding up and allowing us to make greater gains in the future.

Amdahl’s Law

Amdahl’s law can be formulated as follows:

$$S_{\text {latency }}=\frac{1}{(1-p)+\frac{p}{s}}$$

In other words, it predicts the maximum potential speedup (Slatency), given a proportion of the task, p, that will benefit from improved (either more or better) resources, and a parallel speedup factor, s.

To demonstrate, if we can speed up 10% of the task (p=0.1) by a factor of 5 (s=5), we get the following:

$$S_{\text {latency }}=\frac{1}{(1-0.1)+\frac{0.1}{5}} \approx 1.09$$

That’s about an 9% speedup. Eh, fair enough. If we can swing it, sounds good.

However, if we can speed up 90% of the task (p=0.9) by a factor of 5 (s=5), we get the following:

$$S_{\text {latency }}=\frac{1}{(1-0.9)+\frac{0.9}{5}} \approx 3.58$$

That’s roughly a 250% increase! Big enough that it’s actually worth creating twice as much work; it still pays off, assuming the value of the work dwarfs the cost of the resources.

$s \rightarrow \infty$

, which means $\frac{p}{s} \rightarrow 0$

, so we can also drop the $\frac{p}{s}$

term if we can afford potentially infinite resources at no additional cost.

$$S_{\text {latency }}=\frac{1}{1-0.9}=10$$

In other words, if 90% of the work can be parallelised, we can achieve a theoretical maximum speedup of 10x, or a 900% increase. This is highly unlikely, but gives us a useful upper bound to help us identify where the bottleneck lies.

Generalising a PID to the amount of work

Typically, we start off with a completely serial process. In order to parallelize, we need to do more

work. It doesn't come for free.

This means that when computing s

, the parallel speedup

, we should divide it by the cost of parallelisation

.

For example, if the cost is 2,

that means that making the work parallelisable

(without actually increasing the number of resources) makes the parallel portion take twice as long as it used to. (The serial

portion is unchanged.)

So, if we take the example from earlier, where 90% of the work is parallelisable but

it costs twice as much to parallelize, we'll get the following result:

$$S_{\text {latency }}=\frac{1}{(1-0.9)+\frac{0.9}{\frac{5}{2}}} \approx 2.18$$

It's still about a 117%

increase in output! However, if $p=0.1$

, then there's really very little point in adding more resources.

$$S_{\text {latency }}=\frac{1}{(1-0.1)+\frac{0.1}{\frac{5}{2}}} \approx 1.06$$

And if the cost of parallelisation is greater than the potential speedup, bad things happen:

$$S_{\text {latency }}=\frac{1}{(1-0.1)+\frac{0.1}{\frac{5}{20}}} \approx 0.769$$

Adding 4 more resources slows us down by 23%. Many of us have seen this happen in practice with poor parallelization techniques—poor usage of locks, resource contention (especially with regards to I/O), or even redundant work due to mismanaged job distribution.

So, What Does It All Mean?

Amdahl's law tells us something very insightful:

When the value of your work is much greater than the cost, you should optimise for parallelism, not efficiency

.

The cost of a weekly two-hour team meeting is high (typically in the $1000s each time), but if it means that you can have 7 people on the team, not 3, it's often worth it.

Delivering faster means you can deliver more.

Better to have 10 people working on 5 problems and doing a better job than it is to have 10 people working on 10 problems.

The former will lead to fewer conflicts, fewer defects, and a much more motivated team. I.e. p

and s

produce greater returns, faster than the amount of work.

Conversely, if all the knowledge of how the product works is in one person's head, $p \approx 0$

. While there's no impact to efficiency this way, it limits our ability to produce, because one person can only do so much. Adding more people just makes things slower.