2020-07-23

# How can I make my project more environmentally friendly?

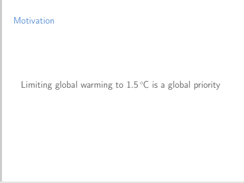
25 minutes allocated. 20 minutes for talk, 5 minutes for questions. Welcome. I'm going to talk about how you can make your project more environmentally friendly. The aims of this talk are to:

- help you to help the environment;
- increase knowledge about where environmental costs lie, and how to avoid them;
- establish an end point for this process: how environmentally friendly do you have to make your project, and how much does it matter; and
- lay out questions which still need answers.

## How can I make my project more environmentally friendly?

Some of you may have been at Aditya's talk on the first day of the conference. If you weren't, I recommend checking out the recording. While his talk looked at helping users identify applications which are consuming power, my talk will look at helping developers reduce the environmental impact of a specific app.

## How can I make my project more environmentally friendly?


Motivation

Limiting global warming to 1.5 °C is a global priority

This is the same slide as in my talk on the environment last year, and it's still as relevant.
Global greenhouse gas emissions need to decline by 45% by 2030 (9 years' time), and reach net zero by 2050 (29 years' time)[**sr15**]. Net zero is where all greenhouse gas emissions that can't be eliminated are balanced by carbon absorption in the environment (by trees) or by using as-yet-nonexistent carbon capture technology. Throughout this talk I'll use carbon dioxide as a proxy for all greenhouse gases. I'll also use energy and carbon interchangeably, as all energy production has a carbon cost of generation (which I'll cover shortly).

The longer it takes for emissions to reach net zero, the higher the global average temperature will rise. So early reductions are better. Typically, warming is expected to be greater than 1.5 °C on land and less than that on oceans[**sr15**].

## How can I make my project more environmentally friendly?

### └Life cycle analysis and products

Life cycle analysis and products

Material Extraction
Manufacturing Production
Transportation
Utilization Reuse
Disposal Recycling

Figure: Life cycle analysis (public domain)

Making and running software emits carbon, and in order to be able to reduce environmental impacts, those emissions need to be measured first. One approach for that comes from the product manufacturing industry: a life cycle analysis (LCA) of the carbon emitted during manufacturing (said to be 'embodied' in the product), during use, and during disposal.

In LCA, a 'functional unit' is defined as a certain quantity of the product being analysed, chosen to make the analysis well-defined, scalable and comparable. For example, the functional unit might be a single car, produced and driven at 6 l/100km for 300 000 km and then scrapped.

A 'system boundary' is defined which includes all raw material inputs and processes which are directly or indirectly needed to manufacture, transport, use and dispose of one functional unit of the product.

2020-07-23

## How can I make my project more environmentally friendly?

### └─Life cycle analysis and products

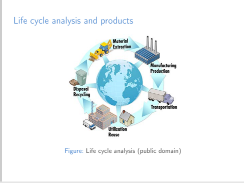

Life cycle analysis and products

Figure: Life cycle analysis (public domain)

In the case of a car, this would include the raw material extraction and processing for the metals, plastics, rubber, etc. in the car; the manufacturing of the car itself, delivery to its owner, the oil extraction and processing for its fuel, the emissions from driving it, and the emissions (or energy and material recovery) from scrapping it and recycling parts of it.

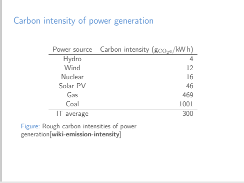
This kind of analysis is well understood (if not universally adopted yet) in the manufacturing industries. It's standardised as the ISO 14000 series.

Life cycle analysis and products

Material Extraction
Manufacturing Production
Transportation
Utilization Reuse
Disposal Recycling

Figure: Life cycle analysis (public domain)

Just like cars, software has embodied carbon costs. Those costs don't come from raw material extraction or burning petrol, but they come from building and powering servers, powering networks for data transfer, and the marginal power usage of the software on an end user's computer (the additional power usage compared to if the user wasn't using that software). About the only part of the lifecycle of software which doesn't emit carbon is disposal.



Carbon intensity of power generation

| Power source | Carbon intensity ($g_{CO_{2}e}$/kW h) |
|---|---|
| Hydro | 4 |
| Wind | 12 |
| Nuclear | 16 |
| Solar PV | 46 |
| Gas | 469 |
| Coal | 1001 |
| IT average | 300 |

Figure: Rough carbon intensities of power generation[**wiki-emission-intensity**]

Most software carbon emissions come from generating the energy needed to power hardware, and hence depend hugely on the carbon intensity of that power generation (coal is carbon intensive, solar is not).

Other emissions come from the embodied costs of the hardware itself — roughly 1.2 $t_{CO_{2}e}$[**goclimate-servers**] to manufacture a 2019 Dell server (not including running, transport or disposal costs). The embodied cost of a laptop is roughly a quarter of that[**connell-2010-laptop**].

Embodied carbon in software

GNOME could provide carbon labelling for what we produce

I've recently bought insulation for my house, and when buying it, you can look at the datasheet and see what its lifetime carbon cost is. You can compare insulation from different manufacturers on this basis.

I would be happy to be corrected, but I know of no analyses of the embodied carbon costs of software.

This is an area where GNOME could improve the state of the software field. I believe people are eventually going to want to know the carbon costs of software they procure (especially at business/government scale). We could provide that information, just like manufacturers of insulation do — and just like we already provide information about software licensing.

In order to do a rigorous analysis, there are complications like direct and indirect rebound effects and transformational changes which would need to be accounted for[**Court_2020**]. I've ignored them for now, for the sake of making some progress.

## How can I make my project more environmentally friendly?

### └Functional unit and system boundary

Functional unit and system boundary

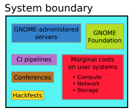


Figure: Proposed system boundary for GNOME

So, let's try and do a rough life cycle analysis for software in GNOME. Here's a functional unit (the thing we want to measure the lifetime costs of), and a system boundary (the processes which we are directly or indirectly responsible for which emit carbon throughout that lifetime).

For certain parts of the system, it might be easier to measure costs in aggregate (for example, the power consumption of all the GNOME servers, or the carbon emissions from running a conference, or the embodied cost of new server hardware).

For other parts of the system, it might be easier to measure costs per functional unit (for example, the resources used for continuous integration (CI) of a particular project, or the emissions from running a hackfest for it).

As long as we're careful to not doubly-count costs, this split should work out.

2020-07-23

## How can I make my project more environmentally friendly?

### └─Functional unit and system boundary

Functional unit and system boundary

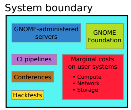


Figure: Proposed system boundary for GNOME

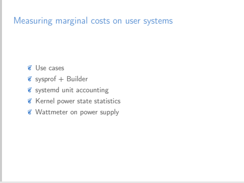
Why is it important to look at the whole system, rather than just running profiling tools on your app? Because looking at the whole system counts the project-wide costs and gives an incentive to reduce them which wouldn't exist otherwise.

That said, let's look first at the marginal costs of an app on user systems.

2020-07-23

How can I make my project more environmentally friendly?

└─Measuring marginal costs on user systems



Measuring marginal costs on user systems

- Use cases
- sysprof + Builder
- systemd unit accounting
- Kernel power state statistics
- Wattmeter on power supply

The marginal costs of your app on a user's system are all the resources it uses while installed or running. Disk space, memory, compute time, network bandwidth, etc. While the improvements in power usage you could make to your app are likely small, they're multiplied by the number of users of the app, which is likely large. I estimate marginal costs like this are the largest environmental impact GNOME has (see my talk last year). There are various conventional tools for measuring and improving these marginal costs — but the first tool you should use is thinking. What use cases is the app for, how does it serve them, and does it serve them in an energy-efficient way?

## Measuring marginal costs on user systems: Use cases

What use cases are you actually solving?

For example, if your use case is allowing the user to listen to music, is it energy-efficient to stream the music from a server on the internet all the time, vs storing it locally?

Or if your use case is for the user to process data in some way, but your software requires the data in a certain format which means that sometimes the user has to spend 30 minutes manually reformatting the data first, then your software is causing 30 minutes of extra computer use (and frustration for the user) due to not implementing all the appropriate functionality. That costs energy.

2020-07-23

How can I make my project more environmentally friendly?

└─Measuring marginal costs on user systems: sysprof + Builder

Measuring marginal costs on user systems: sysprof + Builder

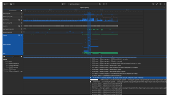
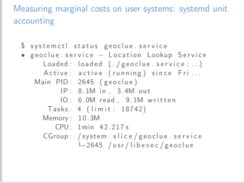


Figure: sysprof results in GNOME Builder

Other tools are available to pinpoint more specific energy consumption issues. Frequent CPU wakeups, network use and disk I/O are the biggest consumers of energy for an app, typically.

1 hour of heavily using the CPU will use about 6 $g_{CO_2e}$ (based on a 20 W swing in power usage and 300 $g_{CO_2e}$/kW h intensity[**Mahesri2004PowerCB**]). 1 GB of network traffic costs about 17 $g_{CO_2e}$[**coroama-hilty-2014**].

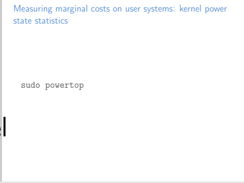
Use sysprof from within GNOME Builder, or `sysprof-cli` from the command line, to plot resource usage on a timeline. There has been recent work in GLib (unreleased) and libsoup (unmerged) to improve reporting of wakeups and network usage.



## Measuring marginal costs on user systems: systemd unit accounting

```
$ systemctl status geoclue.service
• geoclue.service – Location Lookup Service
     Loaded: loaded (../geoclue.service;...)
     Active: active (running) since Fri...
   Main PID: 2645 (geoclue)
         IP: 8.1M in, 3.4M out
         IO: 6.0M read, 9.1M written
      Tasks: 4 (limit: 18742)
     Memory: 10.3M
        CPU: 1min 42.217s
     CGroup: /system.slice/geoclue.service
             └─2645 /usr/libexec/geoclue
```

systemd supports accounting of various resources which processes use, including CPU time, I/O and network bandwidth. It must be turned on, but will then be collected for all units. With the recent work to use systemd for user sessions in GNOME, this gives reasonable (but not total) coverage of session processes, and is a good way to get an at-a-glance look at costs and their longer-term averages over a session.


Measuring marginal costs on user systems: kernel power state statistics

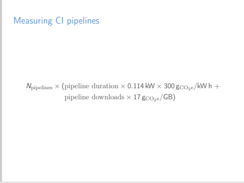
```
sudo powertop
```

`powertop` is an established way of estimating the power consumption of processes and of various bits of hardware, and can provide some insight into the power consumption caused by wakeups from your process. These occur whenever your process wakes up to handle input, timer events, idle callbacks or ongoing disk I/O and network traffic.

As per Aditya's talk at the start of the conference, we may eventually get a breakdown like this within GNOME Usage.

## How can I make my project more environmentally friendly?

### └Measuring CI pipelines



Measuring CI pipelines

$$N_{\text{pipelines}} \times (\text{pipeline duration} \times 0.114\,\text{kW} \times 300\,\text{g}_{\text{CO}_2\text{e}}/\text{kW h} + \text{pipeline downloads} \times 17\,\text{g}_{\text{CO}_2\text{e}}/\text{GB})$$

Moving to look at the embodied cost of a specific release of some software, rather than the marginal cost of running it. CI is an ongoing cost of development. Its main carbon cost is CPU time, but network bandwidth can also end up being significant, especially if every CI pipeline downloads the dependencies for your project from scratch.

The above formula estimates the costs of CI for a project per unit time. You can extract the numbers from GitLab. When I calculated it for GLib, the monthly cost was $4\,\text{kg}_{\text{CO}_2\text{e}}$, which is think is not too high. However, GLib is moderately careful to be efficient with its CI.

For comparison, the target emissions for one person for a year are $4.1\,\text{t}_{\text{CO}_2\text{e}}$[**shrinkthatfootprint**], of which this would be 1.2%.

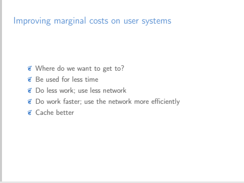
## How can I make my project more environmentally friendly?

### └─Measuring the other bits

We're measuring GUADEC (thanks Bartłomiej!)

Limited work has started on measuring these project-wide overheads, but there is more work still to be done, and there's not enough information available to report in a talk yet.

GUADEC this year is being measured, to provide a comparison against the carbon emissions from an in-person conference. Thanks Bartłomiej for getting the measurements set up! I'll be looking into the results after the conference is over.


Improving marginal costs on user systems

- Where do we want to get to?
- Be used for less time
- Do less work; use less network
- Do work faster; use the network more efficiently
- Cache better

Once measurement is done, improvements can happen. At what point do we stop making improvements? Once an application consumes zero energy? Obviously that's not possible. My current thoughts on this are that it's a competition: if two pieces of software provide the same functionality, rationally users and distros will choose the one with the lower embodied and marginal carbon costs.

These costs should be budgeted for by the user who is running the software — just like the carbon emissions from buying and driving a car should be budgeted for by the driver, and car manufacturers should compete on producing cars with low embodied and marginal emissions.

The improvements you can make to your software are all the standard ones for performance: allow the user to work more efficiently (spending less time on the computer); do less work, use less CPU to do it; use less network bandwidth through compression and caching, and bunching network requests together; and the same for disk I/O.

## Improving CI pipelines

- Speed up your pipelines (use pre-built Docker images)
- Avoid downloads (use pre-built Docker images)
- Use shallow clones (see my blog)

Unless your CI pipeline is run very infrequently (say, a few times a week), you should pre-build a Docker image containing all your dependencies, and run the CI jobs using that image.

That avoids network activity (which is carbon intensive, and can spuriously fail) and speeds up CI jobs due to not waiting for downloads. Typically it will speed up your CI pipeline by a factor of around 4.

Using shallow clones for cloning the git repository into a CI runner will also give some slight speedups, but significant reductions in network use. I wrote about this on my blog recently, so please go there for details — fixing it is a matter of changing a setting in GitLab.

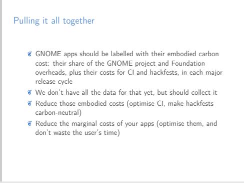
More measuring to do: Foundation operations, and every time a hackfest is organised

We're still in the process of measuring other GNOME project overheads, such as the cost of server infrastructure and the Foundation as an employer and organisation. Once the measurements are done then some recommendations can be made on the basis of the data.

2020-07-23

How can I make my project more environmentally friendly?

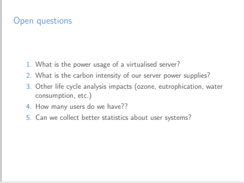
└─Pulling it all together



Pulling it all together

- GNOME apps should be labelled with their embodied carbon cost: their share of the GNOME project and Foundation overheads, plus their costs for CI and hackfests, in each major release cycle
- We don't have all the data for that yet, but should collect it
- Reduce those embodied costs (optimise CI, make hackfests carbon-neutral)
- Reduce the marginal costs of your apps (optimise them, and don't waste the user's time)

So here's how we pull it all together.



Open questions

1. What is the power usage of a virtualised server?
2. What is the carbon intensity of our server power supplies?
3. Other life cycle analysis impacts (ozone, eutrophication, water consumption, etc.)
4. How many users do we have??
5. Can we collect better statistics about user systems?

A lot of this analysis relies on imperfect data. We can't wait for perfect data before making improvements; we should improve the project, software, and data collection in parallel.

If anybody's got any ideas about these questions, please get in touch!

## Miscellany

| | |
|---|---|
| Slide source | `https://gitlab.com/pwithnall/guadec-environmental-presentation-2020` |
| IPCC SR15 summary | `https://www.ipcc.ch/sr15/chapter/spm/` |

And that's it! Please go out there and see what you can do to improve your applications. Mostly, it will be performance improvements and network usage reductions. Sometimes, it might make sense to add or rearrange features to make better use of the user's time. Please get in touch if you have ideas, feedback or want to discuss things further — either about application profiling, or about the climate crisis in general.

If you want to check my analysis, please check out the git repository linked.

The bibliography there has a number of references — if you read just one, read the IPCC's SR15 report summary for policy makers[**sr15**] (it's short).