July 23, 2020 – GUADEC2020

How can I make my project more environmentally friendly?

Philip Withnall

>> For your talk. My talk about how we can make the project more environmentally friendly --

>> Hello, everyone. I'm excited to introduce our next presenter, Philip Withnall. He will be talking about how we can make our project more environmentally friendly. Good luck, Philip.

Philip Withnall: Thanks. So, yeah. Welcome. I'm going to talk to you about how you can make your project more environmentally friendly. The aims of this talk are to help you to help the environment. Include knowledge about where environmental costs lie and how to avoid them. Establish an endpoint to this process. So, how environmentally friendly do you have to make your project and how should it matter? And to lay out questions which still need answers.

Some of you may have been at Aditya's talk on the first day of the conference. I recommend looking at the recording. Helping to look at applications and power on the computer. My talk will talk about helping developers to reduce the environmental impact of a specific application that they work on. So, this is the same slide as in my talk on the environment last year and it's still as relevant, I think. And provides a bit of context and motivation for the entire talk.

Global greenhouse gas emissions need to decline by 45% by 2030. 9 years time. And then to 2050. And net zero, all greenhouse gas emissions are balanced by carbon and by environment, by growing trees or by using as-yet nonexistent carbon capture technology in the air. And using carbon dioxide as a proxy for all greenhouse gases, and use energy and carbon interchangeably. Because most part they are. All energy production has a carbon cost generation and you will cover that shortly.

The longer it takes for emissions to reach net zero, the higher the global average temperature will rise. Kind of like coasting on a road. So, early reductions are better. And typically warming is expected to be greater than 5 degrees C on land and less than that on the oceans.

So, making and running software in less carbon. And be able to reduce environmental impacts. Needs to be measured. That comes from the product manufacturing industry. And that approach is the life cycle analysis of the carbon emitted during manufacture which is said to be embodied in the product, during use and during disposal.

In a functional unit, a well-defined term, the quantity of the product being analyzed chosen to make the analysis well-defined, scalable and comparable between different parts. So, for example, a functional unit in one analysis might be a single car produced and driven 6 liters per 100K for 300,000 kilometers and then scrapped.

A system boundary, another well-defined term. Includes all the raw material inputs and all the processes which are involved directly or indirectly in had manufacturing, transporting, using and disposing of the functional unit. In the case of a car, this would be the raw metal processing for the metals and plastics and rubber and so on in the car. The manufacturing of the car itself. Delivery to its owner. The oil extraction and the processing for its fuel and then the emissions from driving it.

And then the emissions for energy recovery and material recovery from scrapping and recycling whatever you can out of car at the end of its life.

This current analysis is well understood if not universally adopted in the manufacturing industry. It's standardized as the ISO14,000 series. And just like cars, the costs come from raw materials. Or from burning petrol. They come from building and powering servers. So, there is some raw material extraction going on. Powering networks for data transfer and providing the equipment for doing that. The marginal power usage of the software on an end user's computer. Which is the additional power usage compared to if the user wasn't using that software.

And the life cycle of software, which doesn't mimic carbon is disposal of software. Or disposal of service probably as well. So, all that life cycle.

Most carbon emissions come from generating the energy needed to power hardware. And it's on the carbon intensity of that power generation. So, for example, coal, everyone's favorite, is very carbon intensive. You burn coal. You release carbon dioxide. Solar or hydro, much less carbon intensive. But they still have emissions because you need to build with concrete or manufacture and build the factories. So, these carbon intensities factor in the entire supply chain of producing that.

It's quite hard to get figures for the intensity of the electricity you're using. So, I've kind of used an average of 300 grams of CO2 equivalent per kilowatt hour for most of this presentation. Other emissions come from embodied costs for the hardware itself. So, there's roughly 1.2 tons to manufacture a 2019 Dell server. That's not including the cost of running it or disposing of it. And the embodied cost of a laptop is about a quarter of that. So, about 300 kilos of CO2 just to level that top.

I've recently bought insulation for my house. And when buying it, you can look at the data sheet before buying it. And you can see what the lifetime carbon cost of making the insulation is. And then you can calculate carbon savings from not leaking heat from your house into the outside environment. And so, decided to compare insulation from the different manufacturers on the base of the carbon cost. And they do vary radically by an order of 2 orders of magnitude, I think? I would be happy to be corrected. But I know of no signals of the embodied carbon gases of software. Hardware, but not software. So, the scenario, I think, could improve the status of our field. People are going to eventually want to know the carbon cost of software they procure, especially if they're a business or the government and operating at that kind of scale. And we could provide that information. Just like manufacturers of insulation do and just like we already provide information about software licensing. And people already make decisions on that basis.

And in order to do a rigorous analysis, there are various complications like direct and indirect rebound effects and transformational changes which would need to be accounted for. And I've ignored them for now for the sake of making some progress. But hopefully more rigor can be added over time.

So, let's try and do a rough life cycle analysis for software in GNOME. Here's the functional unit that we're gonna use. One distribution tarball of a software release. This is the thing that we want to measure the lifetime costs of. And here's system boundary with some colors. So, system boundary contains the processes which we aren't directly or indirectly responsible for throughout the lifetime of the functional unit. For person parts of the system, it might be easier to measure costs in aggregate. So, for example, the power consumption of GNOME servers or the carbon emissions of the Foundation or conferences.

But for other parts of the system, it might be easier to measure costs per functional unit. For example, the cost of all CI pipelines you run in order to one release of your software. Or the costs of hackfests during your software development to get a certain feature done or something. So, as long as we're careful to not doubly count costs, splitting things up like that should work out okay. 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 produce them which wouldn't exist otherwise. If you only did profiling on your app, you would only be looking at these costs here.

So, that said, let's firstly look at the costs here. The marginal ones. The marginal costs of your app on a user's system are all of the resources it uses while running. That's compute time, bandwidth, that kind of thing. While improvements in power usage you can make to your app are likely quite small, multiply by the users app, likely quite large. Based on the analysis last year, I estimate the marginal costs like this. The largest overall environmental impact that can be in the house. And you want to see more about that, take a look back at my talk from last year.

There are various conventional tools for measuring and improving these marginal costs. But probably the first tool you should use is just to think about your software. What use cases is your app for? How does it serve them? And does it serve them in an energy efficient way?

So, 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, or more energy efficient to store it locally? That's the example of the kind of question that you should be thinking about. And obviously there are other considerations to bear in mind. But it's all about finding a nice balance between them.

Another example, for your use case to process the user in some way. The software requires a certain format. Sometimes the user has to spend half an hour manually reformatting the data before running the software. Then the software has cost 30 minutes of extra computer use and frustration and cost of energy. So, sometimes adding features or making sure that the functionality is appropriate is going to improve the environmental impact of your software.

Other tools are available to pinpoint more specific energy consumption issues. So, frequency of CPU wakeups, network usage, disk IO. They're the biggest consumers of energy in an app. As a rough guideline, one hour of using the CPU will emit about 6 grams of CO2 based on a 20-watt power string. One gigabyte of network traffic will cost you about 17 grams of CO2. And -- and you can explore this by using sysprof-cli on the command line. Or the equivalent. Here is sysprof being used in software. It's the software. It basically plots various different measurements on a timeline. You have timing increasing that way and different measurements of the CPU or the network usage and the bandwidth.

And there's some recent work in globe, and libsoup, which is unreviewed and unmerged but hopefully will go somewhere on reporting globe and others. And the HTTP requests in libsoup.

systemd is another tool you can use. Supports accounting of various resources which processes use. And this is kind of related to Benjamin's talk earlier. It can -- if you enable the accounting functionality by using this command, it will turn on all the accounting for all the units on a system. And with the recent work to use systemd for use, it will get enabled for user session unit as well. It will produce results like this if you're on systemctl status on a service. It will give you an overview of the total network bandwidth that's been used. Or the disk IO bandwidth or the amount of CPU time. And an indication of the runtime as well. So, you can look at systemctl status for your app and find out what it's been doing as a more high-level and over a slightly longer time period of like one whole session or something. Which will give you an idea of where the big costs are.

So, for example, geoclue, not quite sure why it's downloaded 8 megs just to find the location. So, maybe there's a bug there.

At an even higher level, powertop, the various bits of hardware on your machine, and looking at consumption caused by wakeups from your process which add up quite easily to cause a lot of CPU power usage. These occur whenever your process wakes up to handle inputs or timer events or callbacks or disk network traffic and to reduce the power consumption used to group them all together.

And as per Aditya's talk at the start of the conference, we may eventually get a breakdown like is provided but powertop within GNOME usage.

So, moving now to look at the embodied cost of a specific structure like specific costs from your software rather than from-- from a release of your software rather than the marginal costs, sorry. CI, continuous integration, is an ongoing cost of development. And its main carbon cost is CPU time. But network bandwidth adds a bit depending on how you have certain things set up. So, especially if every CI pipeline downloads the dependencies for your project from scratch, that will start to add up to be quite a lot if you run your pipelines quite a lot.

So, the formula here estimates the number of -- the cost of CI for a project over whatever unit of time you set. So, you could say what was the average run for a pipeline in the last month. And how much data was downloaded by each pipeline last month? And then times it by the number of pipelines that happened last month. And you could do that all, and you get numbers like 4 kilos of CO2 out. Which is what I worked out for Glib per month which I think is not too high. It's not zero, but it's reasonable. So, it comes out to four kilos per month. For comparison, the target emissions for one person per year are 1.4 tons, of which that would be 1.2% for doing all of the CI for GLib. And the other bit of the system background are a bit harder to measure and the work is still ongoing on measuring them. There's not enough information to report about in the talk yet. Sorry.

GUADEC this year is being measured. To provide a comparison against carbon emissions from an in-person conference. So, we've got some data from the last two years of GUADEC and hopefully can collect more next year if we do an in-person GUADEC. And we can see how the different options compare and where there are possibilities for savings without affecting the experience of the conference.

Thanks for getting the measurements set up for GUADEC to start this year. I'll be looking at the results. And hopefully publishing something about them after the conference is over.

So, oh, that slide -- once the measurement is done, improvements can happen. And at what point do we stop making improvements once we start it? It's like, do we get to a stage where an application consumes zero energy for doing everything? Obviously that's not possible. So, my current thoughts on this is that it's a bit of a competition. If two softwares provide the same functionality, rationally users and distributions will choose the one with the lower embodies and the lower marginal costs. And I suspect in many cases it won't be that clear cut. But that's the principle. These costs should be budgeted for by the user running the software. Just like the carbon emissions from buying and driving a car should be by the driver. And they can look at producing cars with lower embodiment of carbon emissions. The improvements are the standard ones. Allow them to work more efficiently. Use less CPU to do it, less network through caching and a bunch of network requests together to reduce RAM trips. And do the same for disk IO.

CI pipelines are quite easy to improve. So, unless your CI pipeline is run very infrequently, like a few times a week, then you should pre-build a Docker image containing all of your dependencies and run the CI using that image. That would look at the network activity on each CI job. Quite carbon-intensive. And it can also fail and that's quite often in development. And also means that you don't have to wait for the downloads to happen for all of your dependencies. And why not implement this on various projects? So far, it's sped up the pipeline by about a factor of 4. Using shallow clones for a Git repository into a CI runner. It will reduce the network activity. And that's a simple option you can set in GitHub which I blogged about recently.

So, as I said, we're still in the process of measuring the overheads such as the server infrastructure and the cost of the Foundation as an organization and as an employer. Once the measurements are done, then some recommendations could be made on the basis of the data and we can see how those costs get split up between different projects and within GNOME. And how they compare to the cost of the projects themselves.

So, pulling it all together. GNOME apps, I think, should be labeled with their embodied carbon cost just like they're labeled with a license. And that cost is the their share of the cost of the GNOME project and the Foundation overheads, plus their costs for continuous integration and hackfests that apply to each functional unit, each release of a basic software.

We don't have all the data for that yet. Particularly around project-wide overheads. But we should collect it and we should refine our analysis as more data comes in. Those embodied costs should be reduced based on the data we measure. By optimizing the CI. By making hackfests carbon-neutral. By switching the Foundation servers to be using a carbon-neutral power supply. There's various options. It depends on what the data says.

And we should reduce marginal costs of apps by making them faster in all the normal ways that you optimize software. Then making sure also that I think the key thing is to not have the user use the computer for longer than is necessary. Because it has a certain power needed to be turned on from scratch and then the marginal power increase of the laptop or computer when it's working hard versus idle is a small fraction of that. So, the best thing to do is to not have the computer on in the first place.

There are so many open questions about where the data is inaccurate your, why there are margins? Things we don't know. How many users do we have? Still don't know that one. Can we collect better statistics about the user systems? Which ones are used the most? Which one uses the

CPU the most. We have no framework for that. That would be nice to have. Other life cycle impacts, there are various standard ones. Most of which don't apply to GNOME. We haven't done that at all. There's lots

If anyone has questions about any of these, please get in touch. And that's it. Please go out there and see what you can do to improve your applications, improve your CI pipelines. Mostly I think the improvements will be CPU usage and network usage. I think we're not great on at the moment.

Sometimes it might add or rearrange features to make better use of the user's time. So, please get in touch if you have ideas or feedback or want to discuss things further about the application profiling or about the climate crisis in general and how that applies to GNOME. If you want to check my analysis, please check out Git repository here. And there's a bibliography in there which has a number of references. And if you want to read just one of them, make sure it's that one. It's a 15-minute read. It's really easy. But it covers most of the major topics about climate change.

And that's it. I'll take a look at the questions now. If you have any questions, please put them in Etherpad which is linked in the shared notes. So, first question. have you calculated the carbon cost of something you've made? No. Still a work in progress. Various with the numbers in the presentation are parts of the solution to work out the carbon cost of julip, but I'm in the there yet. There's still thinking to be done about how to attribute the cost of dependencies which I don't have a good answer for yet. It's a work in progress. But we're getting there.

Second question. Are there models from other industries that would show what carbon emission labeling or reporting would look like that we can learn from? Not that I know of. There are standards for carbon emission labeling reporting and there are a lot of different options. And nobody's yet done it successfully in a consumer-facing way. A lot of what's out there at the moment is for business-to-business sales and things. If you've got any ideas of things that do exist and work in certain contexts, let me know. There's also a lot of literature out there and I haven't read it all. I have read quite a lot. But not all of it.

Third question, do you think there is space to create harder policy guidelines down the road? Or even just best practice guidelines? I think there is. I think from the experience after the equivalent talk I did at GUADEC last year, people aren't fans of harder policy guidelines without data to back it up. Which kind of slows down a lot of policy improvement. But also means you don't do the wrong improvements.

So, I would hope there were specs to create harder policy guidelines down the road. But I think we should probably start with softer ones or things we can be fairly confident about. Next question.

If we label apps with the carbon cost, will it be easy to include costs of system services, so, any GeoClue example, or should we also label the core of GNOME separately? Don't know. So, the labeling I'm thinking of is labeling the embodied cost of the release of a piece of software. And I'm not thinking of labeling the marginal costs of running it. So, with that said, I think the question is working on a slightly different basis from what my thoughts were around labeling. But there's possibly code for labeling. Or maybe providing some insight into power usage from certain apps as you run your computer. Which is probably more what -- what people want. Which is what Aditya was talking about in the first talk in the conference. Answering the question of, like, why is my fan running so hard? I didn't think was doing anything really intensive on my laptop. Where is the power going?

Next question. Could someone theoretically --

>> Thank you, for your talk. And we are a little bit out of time.

Philip Withnall: I'll catch people on chat afterwards. Thank you.

>> Thank you very much. Well, thanks, everyone. And you can check break and office hours with GitLab during the break. Good luck.

[Enjoy your break!]