Lineage first computing: towards a frugal userspace for Linux

Michael Dales, Patrick Ferris, Anil Madhavapeddy
University of Cambridge, UK

Data-science is a vital tool in tackling the ongoing climate-crisis, but it is one that has a significant cost to it also, in terms of energy used during execution and in terms of significant hardware investment required to run it. These are then amplified considerably in the exploratory nature of scientific research.We argue this is because the affordances of the operating system, particularly filesystems, make little effort to support reuse of computed artifacts directly or promote reuse through trusted lineage data.

However, it turns out that many of the userspace interfaces exposed by the Linux kernel entirely support a radically more efficient – a more *frugal* – model of computation than is currently the default. In this talk, we explore a new OS architecture which defaults to deterministic, reusable computation with the careful recording of side effects. This in turn allows the OS to guide complex computations towards previously acquired intermediate results, but still allowing for recomputation when required.

We use these interfaces to build a new Linux userspace where we put the workflow graph—containing relationships between tools, provenance and labelling—as the core of a system that drives how processes, data, and users interact. Our prototype shell, dubbed Shark, is a data-science first operating environment that is designed to both ensure efficient use of computation and storage resources, and to make it easy for non-experts to create pipeline descriptions from end results post-hoc. It does this by making the lineage graph of a data-pipeline the key concept that ties everything else together: by tracking how the data-pipeline evolves from experimental practice, and tracking what data has already been built and what hasn't we can prevent re-execution both at development and after publication.

1 WASTE IN DATA-SCIENCE

Whilst it is not intrinsic to the domain, we observe that in practice data-science is often wasteful of resources, driven in a large part by *uncertainty* in the process [5]: uncertainty in data (e.g., which datasets and versions have been used in the past), uncertainty in code (e.g., who ran which version of a stage), and uncertainty in the methodology (e.g., which existing published results can be verified and trusted in the future). The typical solution to resolve this uncertainty is expensive re-computation, which causes excess scope 2 and 3 emissions [6].

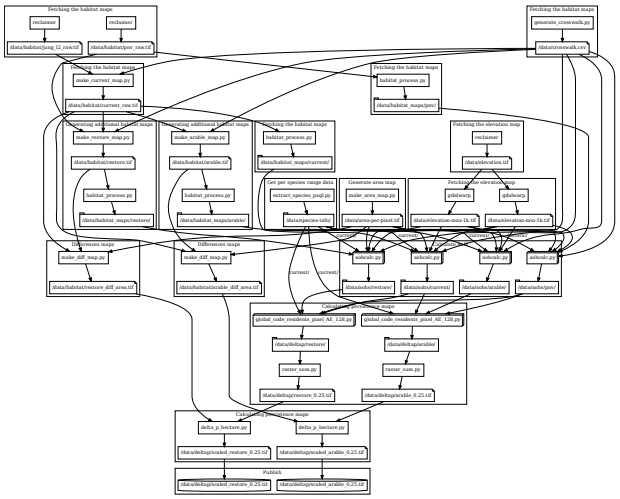

Figure 1: The LIFE biodiversity metric pipeline, showing the processing stages and intermediary datasets required. This pipeline is run for multiple scenarios and results in around 92 petabytes of image data.

We have seen this first hand being embedded in two large ecology projects at the University of Cambridge: the LIFE biodiversity metric [4] and the PACT forest assessment methodology [3]. Both pipelines are large multi-stage workflows like that shown in Figure 1, where the state of the computing is larger than a single person's working set and is distributed across multiple people with different specialities. Combined with the need to process petabytes of data and stages that can take weeks to run, this is a recipe for significant waste.

The current common set of OSs offer no way to query the lineage graph of results, and so it is easier to re-run a pipeline than investigate what went before, leading to wastage.

2 AUTOMATED DATA LINEAGE

A data lineage graph [1], that is a DAG of all inputs and processing stages in a pipeline, serves two purposes for us: it provides a way to attest what happened to humans, and it provides a way to a system run time to both execute the pipeline and to ensure parts already run are not run again.

Whilst data-scientists could manually build lineage graphs, these domain experts are typically focused on the primary

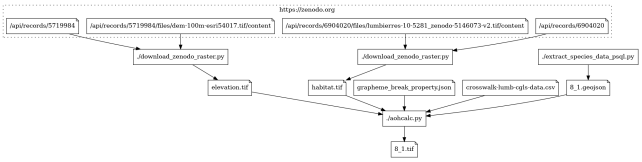

Figure 2: This graph was automatically extracted from final output of unmodified python scripts that form a pipeline to calculate species Area of Habitat.

science mission, and the art of what is possible isn't obvious [8]. Instead, we believe it should be the system's job to capture this as a pipeline like that shown in Figure 1 is built up.

As a demonstration of how readily this can be done, the graph shown in Figure 2 shows the execution trace of a section of the LIFE pipeline being automatically extracted from a series of unmodified Python scripts, by injecting code into their environment, swizzling library calls and then writing a manifest alongside each output file, which is then picked up by later stages and propagated until the final manifest has a record of all the downstream inputs that built that file. No behaviour change was needed by the data-scientists, to result in a manifest that documents primary source information (e.g., downloads from Zenodo), code used (with git commit IDs), lists of libraries imported, etc.

Whilst this demonstration only works for Python, the principles in general can be applied to general purpose program running in a Linux environment using eBPF [2] which allows monitoring of syscalls made, and by pushing network requests through a proxy.

Attestation to humans is made possible by introducing digital signatures at each stage, with the possibility that an institution will sign the final result before publishing.

3 SHARK: A LINEAGE FIRST, FRUGAL DATA-SCIENCE PLATFORM

To push these ideas further, we are building Shark, a data-science environment that puts data lineage first and ensures frugal computation by only executing what it knows not to be already done. This isn't merely done as a post-hoc artifact at the end of the process, rather this principle carries through from initial use to published result.

Shark's aim is to record the data-science pipeline at the command level, removing the requirement that data-scientists learn new languages or libraries to complete their work. These commands are executed using container environments appropriate to each command to allow for dependency tracking, and use ZFS datasets for storage, to allow for snapshotting and build incremental graphs of the pipeline development.

Shark has two key modes of operating: a batch mode and an interactive mode. In the batch mode we take pipeline description that is markdown document with a series of code blocks that describe the pipeline. Using markdown this way is a form of *literate programming* [7] that allows the methodology of the pipeline to be described alongside the commands needed to run it, and because the commands don't rely on Shark directly, this syntax allows others to hand execute the pipeline independently should they so wish. Each code-block has a tag that indicates a particular container environment to be used, allowing a mixing of say R and Python or whatever other tools that data-scientists are familiar with. The Shark markdown files also have special import clauses that facilitate the description of primary resources from locations such as Zenodo or GitHub, encouraging the use of original resources rather than local copies with no history.

As the pipeline is updated and re-run during its development, because Shark knows which containers were used and the file-system snapshots involved, it can hash these to understand if it needs to re-run anything or whether it can use existing results, thus saving the data-scientist time and reducing emissions.

In the code-blocks we also allow an extended shell syntax via wild-carding that allow the expression of map/reduce style stages where Shark can automate parallel execution of jobs based on earlier results in the pipeline, where in the command line world tools like GNU parallel would be used.

In the interactive mode, Shark operates like any other interactive shell, offering a command prompt with access to commands, but as commands are executed, they are split into those that inspect data, and those that mutate data. If data mutation is detected, then Shark will snapshot the result, building up the lineage graph as it goes. At the end of the interactive session that graph can then be used to find which stages contributed to the final result, and that graph can be used as the basis for generating the template Shark markdown document used for future runs.

4 SUMMARY

Uncertainty in data-science leads to a significant wastage of energy via scope 2 and scope 3 emissions. We are building Shark, an inherently frugal environment for data-science that fosters efficient behaviour by putting automated data lineage at the core of the computing process, removing that uncertainty, and thus removing the wastage it causes. Conveniently, it turns out that the Linux kernel already supports many of the features that we need to build such an environment, meaning that our system is compatible with existing cloud computing infrastructure.

REFERENCES

[1] Data lineage. https://en.wikipedia.org/wiki/Data_lineage.

[2] What is eBPF. https://ebpf.io/what-is-ebpf/.
[3] Balmford, A., Coomes, D., Dales, M., Ferris, P., Hartup, J., Jaffer, S., Keshav, S., Lam, M., Madhavapeddy, A., Message, R., Rau, E.-P., Swinfield, T., Wheeler, C., and Williams, A. Pact tropical moist forest accreditation methodology v2.1. Tech. rep., Cambridge Open Engage, 2024.
[4] Eyres, A., Ball, T., Dales, M., Swinfield, T., Arnell, A., Baisero, D., Durán, A. P., Green, J., Green, R. E., Madhavapeddy, A., and Balmford, A. LIFE: A metric for quantitatively mapping the impact of land-cover change on global extinctions. Tech. rep., Cambridge Open Engage, 2024.
[5] Ferris, P., Dales, M., Swinfield, T., Jaffer, S., Keshav, S., and Madhavapeddy, A. Uncertainty at scale: how CS hinders climate research. *Undone Computer Science* (2024).
[6] GHG Protocol. Coprorate standard. Tech. rep., GHG Protocol, 2015.
[7] Knuth, D. E. Literate Programming. *The Computer Journal 27*, 2 (01 1984), 97–111.
[8] Shaw, M. Myths and mythconceptions: what does it mean to be a programming language, anyhow? *Proceedings of the ACM on Programming Languages 4*, 234 (2020), 1–44.