EPJ Web of Conferences **245**, 05025 (2020) https://doi.org/10.1051/epjconf/202024505025
*CHEP 2019*

# Design Pattern for Analysis Automation on Distributed Resources using Luigi Analysis Workflows

*Marcel* Rieger[1,*]

[1]CERN, Geneva, Switzerland

**Abstract.** In particle physics, workflow management systems are primarily used as tailored solutions in dedicated areas such as Monte Carlo event generation. However, physicists performing data analyses are usually required to steer their individual workflows manually, which is time-consuming and often leads to undocumented relations between particular workloads. We present the Luigi Analysis Workflows (Law) Python package, which is based on the open-source pipelining tool Luigi, originally developed by Spotify. It establishes a generic design pattern for analyses of arbitrary scale and complexity, and shifts the focus from executing to defining the analysis logic. Law provides the building blocks to seamlessly integrate interchangeable remote resources without, however, limiting itself to a specific choice of infrastructure. In particular, it encourages and enables the separation of analysis algorithms on the one hand, and run locations, storage locations, and software environments on the other hand. To cope with the sophisticated demands of end-to-end HEP analyses, Law supports job execution on WLCG infrastructure (ARC, gLite) as well as on local computing clusters (HTCondor, LSF), remote file access via most common protocols through the GFAL2 library, and an environment sandboxing mechanism with support for Docker and Singularity containers. Moreover, the novel approach ultimately aims for analysis preservation out-of-the-box. Law is entirely experiment independent and developed open-source. It is successfully used in $t\bar{t}H$ cross section measurements and searches for di-Higgs boson production with the CMS experiment.

## 1 Introduction

The management of scientific workflows presents a complex challenge in today's physics working environments. High-level physics analyses usually consist of a considerable amount of logically separated workloads. In general, the interface between these workloads does not rely on an event-by-event data flow. They rather form a loose collection of inhomogeneous procedures, encoded in executable files such as Shell and Python scripts, and are executed manually. Hereby, their execution order is dictated by the dependencies between them in terms of the existence of results of one or more previous procedures. Beyond a certain scale and complexity, i.e., degree of granularity and inhomogeneity of workloads, manual steering of analysis workflows can be time-consuming, prone to errors, and potentially leads to undocumented relations between workloads.

*e-mail: marcel.rieger@cern.ch

A novel design pattern for physics analyses conception and automation that copes with the challenges of inhomogeneous workload definition and risks due to manual steering is presented. It is based on the pipelining package Luigi [1] due to its simple yet scalable and extensible design, providing guidance on structuring arbitrary workloads. To meet the sophisticated demands of performance-intensive analyses, the developed software, called *Luigi Analysis Workflows* (Law), allows to seamlessly integrate remote resources as used in the high-energy physics community. Conceptually, Law is independent of any experiment, specific use case, programming language, and data format. In typical high-energy physics nomenclature, it does not qualify as a "framework", but it rather defines a set of permissive guidelines and tools to follow a design pattern for distributed analyses. The eventual goal of the software is to provide a versatile working environment for robust analyses on scalable and interchangeable resources, while encouraging analysis preservation.

Specially tailored solutions already exist for a small number of particle physics applications such as the comprehensive event simulation campaigns performed centrally by the CMS and ATLAS experiments at the LHC [2, 3]. However, the requirement profile for performing end-to-end physics analyses can differ greatly. Particular workloads may require the results of multiple previously executed workloads and produce more than one output themselves. These dependencies form a directed acyclic graph (DAG), which can take arbitrary shapes that could even depend on dynamic run-time conditions. In addition, the shape of the DAG is not necessarily known a-priori and also might be subject to continuous development cycles. Furthermore, whereas central experiment workflows often rely on dedicated infrastructure, analyses must rather incorporate existing resources and maintain the ability to adapt to short-term changes.

## 2 Guidelines

The key operating principles of the presented software and the associated analysis design pattern are based on four guidelines, which are discussed in the following paragraphs. A supporting illustration is presented in Fig. 1.

**1. Encapsulation** Logically separated steps of an analysis are encapsulated in workloads. They can produce outputs, require the completion of certain other workloads, and therefore consider their outputs as potential inputs. Workloads and the dependencies between them compose a workflow that is visualizable as a DAG. The graph structure can be exploited by execution systems to process entire workflows using a single command invocation.

**2. Factorization** A workload is factorized into four domains that are organized in two layers. The "management" layer handles the run location of a workload, defines the location where output data is stored, and steers the software environment during execution. The "analysis" layer contains the actual algorithm code to run. It is fully decoupled from the management layer, i.e., algorithms remain unaffected by any decisions about run and storage locations. Dynamic behavior of both layers can be achieved by including parameters to the workload. Their values might be configured at execution-time or depend on run-time conditions.

**3. Interchangeability** Implementations of domains of the management layer are interchangeable. For instance, the change of the storage location neither affects the analysis algorithm, nor decisions about its run location or software environment. Evidently, exceptions occur in cases where special hardware requirements (e.g. GPUs for machine learning), or strict dependencies between run and storage locations exist (e.g. data access on remote computing elements behind firewalls is often restricted to dedicated storage systems). The interchangeability of resources prevents hard-coded dependence on certain infrastructure.

**4. Universality** The realization of algorithms in the analysis layer and the type, quantity, and content of their results are universal. There is no limitation on the programming language or implementation details of algorithms, as long as they are executable by means of sub-processing. Furthermore, the form of analysis results is entirely unrestricted, i.e., any kind of stateful information can serve as workload output. This includes files with arbitrary data formats, database entries, or even specific file content, e.g. contained in machine-parsable job reports. Universality causes the distinction from the concept of "frameworks" in the sense of typical particle physics applications.

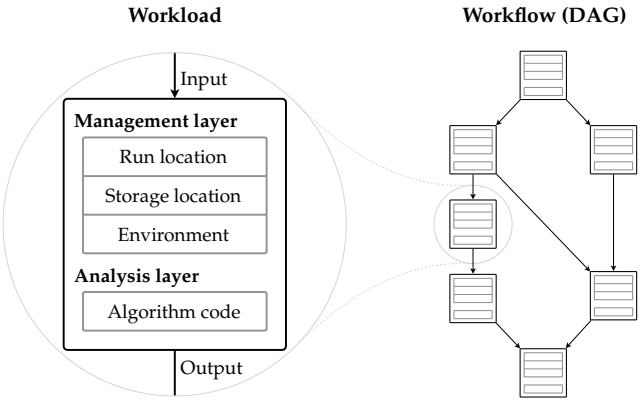


Figure 1: Illustration of a workflow composed of multiple workloads, forming a directed acyclic graph (DAG). The management layer defines and handles run location, storage location, and environment. The analysis layer comprises the actual algorithm to run and is fully decoupled from the management layer, allowing for interchangeability of resources.

Compliance with the four guidelines above entails several accomplishments. Relations between workloads are encoded as dependencies and become an integral part of an analysis. Besides the evident advantages for automation, this also leads to notable benefits for the purpose of documentation. Final and intermediate results are both reproducible and reusable between similar analyses, effectively lowering the degree of unnecessary redundancy.

Technically, the these concepts and guidelines are realizable using the scalable and extensible structure provided by the Luigi package, which is introduced in the following.

## 3 Luigi

Luigi is a Python software package that provides a scalable design pattern for structuring large and complex workflows [1]. Initially developed at Spotify, it became a community-driven, open-source project and is successfully deployed in both commercial and scientific applications. The execution model follows a `make`-like approach as it only computes what is really necessary in order to produce the output of a requested workload [4]. Its features include automatic failure handling, command-line interfaces, web visualization, and file system abstractions. The following paragraphs introduce the fundamental building blocks.

In Luigi, an arbitrary workload, i.e., the elementary unit in an overarching workflow, is described as a "task". The purpose of a task is to produce a customizable set of outputs, denoted as "targets". While targets usually represent local or remote files, they can, in principle, describe any type of stateful resource. The sole core functionality of a target is to check

and report its own existence. Therefore, a task is considered "complete" when all its output targets exist. Moreover, to alter the default behavior of a workload, tasks can expose and implement mandatory and optional "parameters". Finally, tasks can "require" one or multiple other tasks to define a coherent workflow, where outputs of required tasks become their "input". As tasks can implement requirements depending on values of particular parameters, workflows can behave highly dynamically and adapt to a large variety of use cases.

A workflow that consists of interdependent tasks can be described as a directed acyclic graph (DAG), and visualized upon execution using a web application provided by Luigi. An example is shown in Fig. 2 for a hypothetical analysis. The execution of a workflow is initiated by running a particular task. Luigi recursively evaluates the completeness condition of all required tasks to infer the shape of the underlying DAG. Subsequently, tasks are scheduled according to their position in the DAG. As a result, Luigi's stateful execution system is deterministic and resource-effective since it only processes what is really necessary.

For physics applications, workflow management tools such as Luigi can give rise to considerable advantages. Besides the separation of workloads and a consistent parameterization approach, all dependencies are expressed as part of the analysis with direct benefits for error prevention, documentation, and collaboration. At the same time, neither constraints on the programming language nor on the format of data exchanged between tasks is introduced.

## 4 Law

The Law software package extends Luigi to include remote resources available in the context of high-energy physics research. It is designed as a layer on top of Luigi rather than as a replacement, without changing any of its core functionality or depending on a specific version. The guidelines introduced in Section 2 are realized through a set of technological key concepts, namely remote execution of tasks, remote storage of output targets, and software environment sandboxing. In the subsequent paragraphs, they are discussed in detail, followed by a discussion on the entailed benefits for analysis preservation.

*Remote Execution*

Physics analyses investigating rare processes in high-energy particle collisions typically examine a considerable amount of data to infer statistically significant results. This implies that a large quantity of computing tasks must be executed, which most likely exceed the capacity of local host systems. Consequently, remote task execution on distributed batch systems is

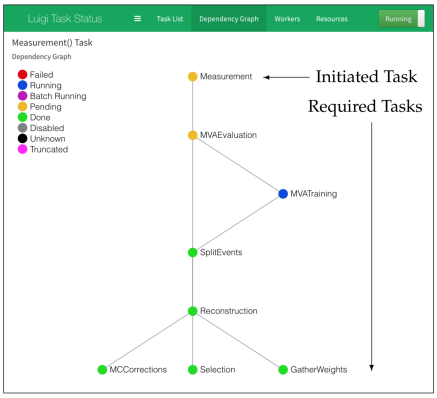


Figure 2: Screenshot of the Luigi web visualizer application, showing a hypothetical analysis workflow consisting of an initiated task and its recursive requirements in a directed acyclic graph (DAG). Edges visualize direct dependencies and node colors denote task statuses.

necessary. Following the principle of factorization as introduced above, analysis algorithm code should remain unaffected by any decisions about the run location.

The developed software realizes remote execution capabilities through "mixin-based" inheritance. Tasks can inherit from one ore more classes that provide functionality for submitting computing jobs to certain remote batch systems. In contrast to inheritance models that denote "is-a" relationships, mixin-based inheritance is considered as a technique to adapt common functionality while avoiding code redundancy. Thus, the configuration of run locations is rather declarative. In case of multiple available run locations, the actual implementation for a particular batch system can be selected at execution time using task parameters, which fulfills the demand for interchangeability. At the time of writing, the HTCondor [5, 6], LSF [7], ARC [8], and gLite [9] batch job interfaces are supported.

*Remote Storage*

The large amount of data to be processed inevitably causes the need for distributed data storage systems. Requirements on the storage capacity per analysis in the order of $\sim 10 - 100\,\mathrm{TB}$ are not unusual. Especially when relying on large-scale remote execution systems such as the WLCG [10], the use of high-throughput storage is mandatory.

The concept of output targets represents a suitable abstraction for handling files on interchangeable remote resources. Independent of whether targets denote local or remote files, they strictly implement the same common interface, which defines both low- and high-level file operations. Algorithms in the analysis layer can rely on the equivalence of operations, fully omitting distinction of cases to achieve interchangeability of storage locations.

The target interface in Law is based on the open-source Grid File Access Library (GFAL2) [11], which enables remote file access through various protocols such as provided by XRootD, dCache, SRM, GridFTP, and Amazon S3. Moreover, the developed interface provides essential features including transfer validation, automatic retries for robustness against connection disruptions, and optional local caching. The latter is of particular importance for the optimization of network utilization in a variety of use cases such as repeated local tests. Both cache synchronization and invalidation are fully integrated into all remote target operations.

*Environment Sandboxing*

Different workloads within the same analysis workflow may depend on distinct software setups that are not necessarily compatible. Examples are workloads that utilize experiment-specific software that is often subject to extensive validation procedures and therefore requires long-term stability. This is opposed to machine learning algorithms built on top of third-party libraries, which may crucially improve in performance when updated on a regular basis. For this purpose, environment "sandboxing" is integrated into Law.

A sandbox describes a collection of environment configurations, ranging from plain sets of variables over to different operating systems. Current implementations support subshells with custom initialization files and virtual environments (VEs), realized through containerization using Docker [12] or Singularity [13]. Tasks can declare the demand for a particular sandbox and, optionally, provide further logic for negotiating with the current environment to determine if either compatibility is sufficient or sandboxing is necessary. In the latter case, the developed mechanism embeds the invocation of the algorithm code within the specified sandbox. This approach preserves the encapsulation paradigm by considering environment dependencies as part of the management layer of workloads and thus, enables the execution of inhomogeneous workflows with a single command.

Another benefit of sandboxing is related to long-term stability and portability. Most host systems and computing clusters are subject to constant changes owing to maintenance and security measures. For workloads with critical software dependencies, sandboxing provides a method for ensuring reproducibility of results over time. Therefore, it exhibits a key technique for the preservation of analyses, which is discussed in the following section.

*Analysis Preservation*

The preservation of analyses is a key ingredient for sustainability of experimental physics research. Long-term storage of data, analysis algorithms, and related external information is required to ensure the repetition of measurements with reproducible results for the purposes of documentation, education, and reinterpretation. Especially reinterpretation campaigns represent an important use case as they could utilize preserved analyses to test hypotheses postulated by newly developed theories, exploiting the amount of recorded data and developed algorithms to this date. Inter-experimental collaborations have formed to advance this common effort [14]. In practice, however, the process proves to be challenging owing to the fact that analyses need to be adapted manually to fulfill certain preservation conditions. In contrast, analysis workflow management systems with integrated preservation capabilities could offer appealing benefits. On the basis of the four guidelines introduced in Section 2, a decisive set of preservation conditions can be derived:

1. Analysis layers of workloads containing algorithm implementations must not be changed. Operations relying on random number generators should use fixed seeds.
2. Software and other relevant environment configurations must be retained such that repeated execution of the analysis code leads to identical results.
3. All initial and external files must either be retained or referenced to copies on dedicated data preservation systems.
4. Access to remote infrastructures providing parallel execution and storage systems must be configurable for analyses that require a considerable amount of resources.
5. The execution of analysis workflows must be fully automated and / or sufficiently documented in order to guarantee operability by other researchers at any time.

Law intrinsically satisfies conditions 1, 4 and 5 by adhering to the principles of workload domain factorization and interchangeability of run and storage locations. The retention of environment configurations (condition 2) can be achieved by means of the sandboxing mechanism. For high-energy physics experiments, the preservation of simulated and recorded collision events (condition 3) requires sophisticated measures. Integration with LHC open-data initiatives [15] is mandatory and therefore encouraged by Law.

## 5 Conclusions

The presented guidelines and tools for generic analyses conception constitute a novel approach for coping with the increasing demands of modern high-energy physics data analysis. The Luigi package is a viable solution to address the complexity of structuring and executing workloads in a `make`-like fashion. The developed Law package adds scalability in the scope of high-energy physics infrastructure in a non-intrusive way by interfacing common job submission systems and remote data storage software. In addition, a customizable sandboxing mechanism ensures the integrity of software and computing environments, and thus the reproducibility of physics results.

As the described approach does not introduce constraints on software or data structures, it is considered a toolbox providing an "analysis design pattern" rather than a "framework". The resulting workflows preserve all information about entangled analysis workloads with benefits for automation, error prevention, and documentation. In a broader context, the presented project provides the means to extend the concept of collaboration beyond the sharing of code. Eventually, the resulting increase of transparency and reproducibility paves the way for analysis preservation.

The Law package was successfully employed in a $t\bar{t}H$ cross section measurement with the CMS experiment where it achieved full automation over distributed resources [16]. The source code is publicly available under BSD license [17].

## References

[1] The Luigi Authors, "Luigi Documentation", 2018, https://luigi.readthedocs.io.

[2] The CMS Collaboration, "CMS computing : Technical Design Report", 2005, cds:838359.

[3] The ATLAS Collaboration, "ATLAS Computing : Technical Design Report", 2005, cds:837738.

[4] Free Software Foundation, "GNU Make", 2016, https://www.gnu.org/software/make.

[5] T. Tannenbaum, D. Wright, K. Miller and M. Livny, "Beowulf Cluster Computing with Linux: Condor - A Distributed Job Scheduler", *MIT Press*, Cambridge, MA, USA, 2002, ISBN: 0262692740.

[6] D. Thain, T. Tannenbaum and M. Livny, "Distributed Computing in Practice: The Condor Experience", *Concurrency and Computation: Practice and Experience* **17** (2005) 323.

[7] I. Lumb and C. Smith, "Grid resource management: Scheduling Attributes and Platform LSF", *Kluwer Academic Publishers*, Norwell, MA, USA, 2004, ISBN: 1402075758.

[8] M. Ellert *et al.*, "Advanced Resource Connector middleware for lightweight computational Grids", *Future Gener. Comput. Syst.* **23** (2007) 219.

[9] P. Andreetto *et al.*, "The gLite workload management system", *J. Phys.: Conf. Ser.* **119** (2008) 062007.

[10] The CMS Collaboration, "LHC computing Grid : Technical Design Report", 2005, cds:840543.

[11] A. A. Ayllon *et al.*, "Making the most of cloud storage - a toolkit for exploitation by WLCG experiments", *J. Phys.: Conf. Ser.* **898** (2017) 062027, cds:2297053.

[12] C. Boettiger, "An introduction to Docker for reproducible research, with examples from the R environment", *ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts* **49** (2015) 71, arXiv:1410.0846.

[13] G. M. Kurtzer, V. Sochat and M. W. Bauer, "Singularity: Scientific containers for mobility of compute", *PLoS One* **12** (2017) e0177459.

[14] The DPHEP Collaboration, "Status Report of the DPHEP Study Group: Towards a Global Effort for Sustainable Data Preservation in High Energy Physics", 2012, arXiv:1205.4667.

[15] J. Cowton *et al.*, "Open Data and Data Analysis Preservation Services for LHC Experiments", *J. Phys.: Conf. Ser.* **664** (2015) 032030, cds:2134548.

[16] CMS Collaboration, "Search for $t\bar{t}H$ production in the $H \to b\bar{b}$ decay channel with leptonic $t\bar{t}$ decays in proton-proton collisions at $\sqrt{s} = 13\,\text{TeV}$", *JHEP* **03** (2019) 026, arXiv:1804.03682.

[17] M. Rieger, "Luigi Analysis Workflows Project", 2018, https://github.com/riga/law.