Preliminary draft 16:12 31 August 2023

31 August 2023
cameron474747@gmail.com

lb-telemetry: Identifying optimisation opportunities in LHCb software tools

Cameron McClymont
Supervisors: Daniel Cervenkov, Chris Burr

Keywords: `Telemetry, Python, MONIT, InfluxDB, Grafana, Rust, HTTP server`

Summary

More time to gather data is required before optimisation analysis can begin, but many other useful results were obtained in the meantime. All statements below refer to internal CERN users only.

PIDCalib2:

- Over the summer, the `make_eff_hists` module was invoked around 200 times per week
- The `sample` is the biggest factor in the module's running time (the Turbo sample increasing it 10-fold on average)
- The module's execution time averages 7 mins and adds to 117 hours of total running time over the summer so it is well worth exploring optimisation strategies

LbConda:

- There are around 8000 activations per week
- The most popular configuration by far is:
  - **OS**: glibc 2.17
  - **Python**: CPython 3.9.16

LbEnv:

- The `/usr/bin/bash` shell is by far the most popular, being used in over 99% of instances
- Only 1% of users use a non-stable flavour (unstable or testing)
- On average, there are about 10,000 total activations per day
- Broadwell and Skylake chips are about equally popular processors, but together are used in over 99% of all activations

Contents

1 Introduction

The LHCb experiment uses a wide variety of software tools (referred to as "target packages") to execute its physics program. Many of the tools use considerable computing resources, especially when repeatedly run by hundreds of users. We expect that a significant part of the users run some of the tools with identical parameters, leading to repeated work and wasted resources. Understanding how the tools are used could allow us to optimise or cache data for some of the common use cases. This project aims to develop a Python package that collects anonymised telemetry and propagates it to CERN's MONIT service.

The project was later extended to also collect information about which software people are using so that maintenance efforts can be targeted more effectively. For example, if nobody uses a particular LbConda environment, there's no need to keep updating it.

2 Target packages

2.1 PIDCalib2

PID calibration is the process of computing the efficiency of particle identification selection requirements. This means quantifying the probability of correctly identifying a particular particle of interest. PIDCalib2 (paper here) is a tool widely used within LHCb that assists with this. It was our first target package, specifically the `make_eff_hists` module.

Interesting metrics to track here included various configuration parameters such as the `binning_file`, `magnet`, `particle`, etc as well as some more general data like `execution_time` and `version`. A payload example is shown below:

```
payload = {
  version:  "1.0.0",
  exec_time:  5737.483985,
  sample:  "Turbo16",
  magnet:  "up",
  ...
}
```

Viewing this data on Grafana, we can see how execution time varies depending on the value set for the magnet. We can also see a table showing how often the package is used and various details about the configuration used.

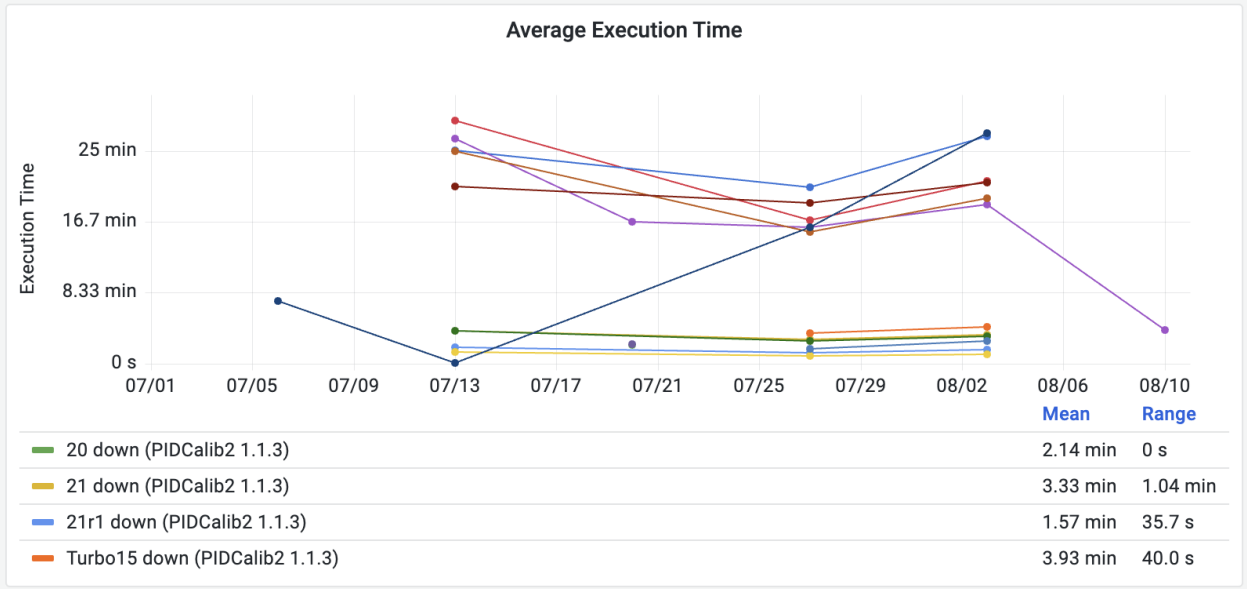

Figure 1: The average execution time of PIDCalib2's `make_eff_hists` module, grouped by version, magnet, sample, and aggregated into weeks.

Recent Popular Args

Time	exec_time	args
2023-07-13 12:40:43.424	29.7 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-13 12:08:07.163	27.5 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-13 09:11:48.981	27.3 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-13 10:47:55.993	26.0 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-13 11:39:50.827	25.8 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-14 18:09:17.868	25.7 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
2023-07-13 11:13:10.827	24.3 min	{"binning_file": "/afs/cern.ch/work/j/jherd/ewp-rk-rkst-highq2/rk/dat...
Mean	14.3 min	

Figure 2: Specific runs of the PIDCalib2's `make_eff_hists` module and their various configurations.

2.2 LbCondaWrappers

LHCb Conda Wrappers provides access to Conda environments installed on CVMFS. It is of interest for us to track the most popular OS/Python versions in use as well as commonly used environments and package configurations. Integration with the package simply involved adding a few lines of Python via a merge request which are triggered when the `LbCondaWrappers` CLI is used. A payload example is shown below:

```
payload = {
  conda_subdir:  "linux-64",
  env_name:  "default",
  env_version:  "2023-07-30_07-32",
  latest:  False,
  with_texlive:  False,
  env_type:  "release",
  dev_id:  "none",
}
```

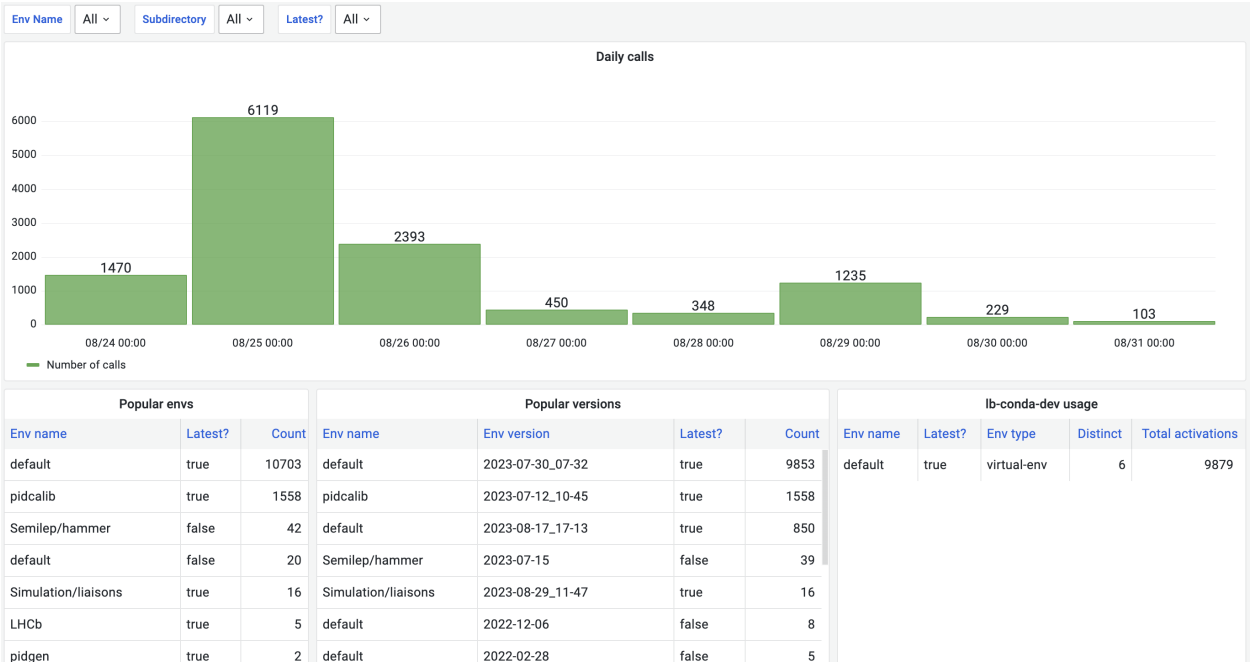

Figure 3: The LbCondaWrappers dashboard.

2.3 LbEnv

In order to collect a broad range of telemetry for all LHCb software, our next milestone was integration with the LbEnv user environment entry point. This involved modifying template scripts installed with LHCb software and required the development of a simple CLI to use in these scripts.

This integration enabled the tracking of LbEnv activations by flavour/build number as well as the types of shell, OS, and processors used. A payload example is shown below:

```
payload = {
```

```
  build_num:  "2876",
  host_os:  "linux-64",
  flavour:  "stable",
  override_lbenvroot:  "false",
  lbenv_prefix:  "/cvmfs/lhcb.cern.ch/lib/var/lib/LbEnv/2876/stable/linux-64",
  shell:  "/usr/bin/tcsh"
}
```

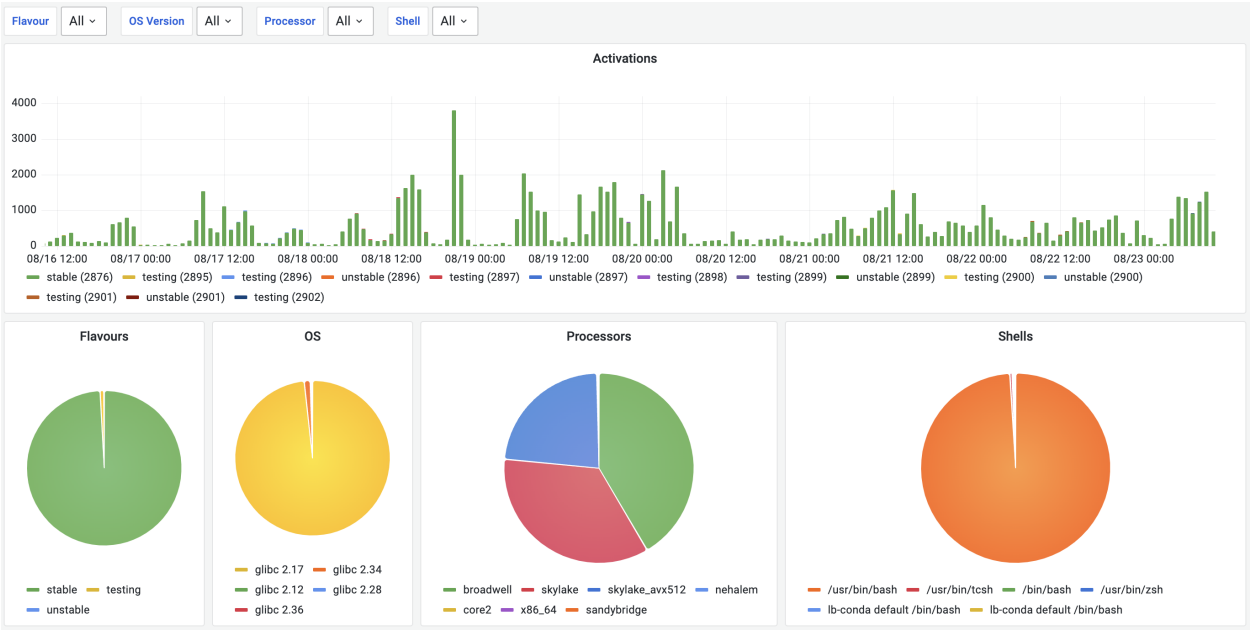

Figure 4: The LbEnv dashboard.

2.4 Future packages

Other packages that may be of interest to investigate in the future include:

- **Nightly environments used**
- **lb-dev**
- **lb-run**
- **ci-tests**
- **Stack builds**

This is beyond the scope of my work, but its generic and maintainable design will make it easy for future developers to expand its reach to these packages.

3 Design

3.1 Flow of data

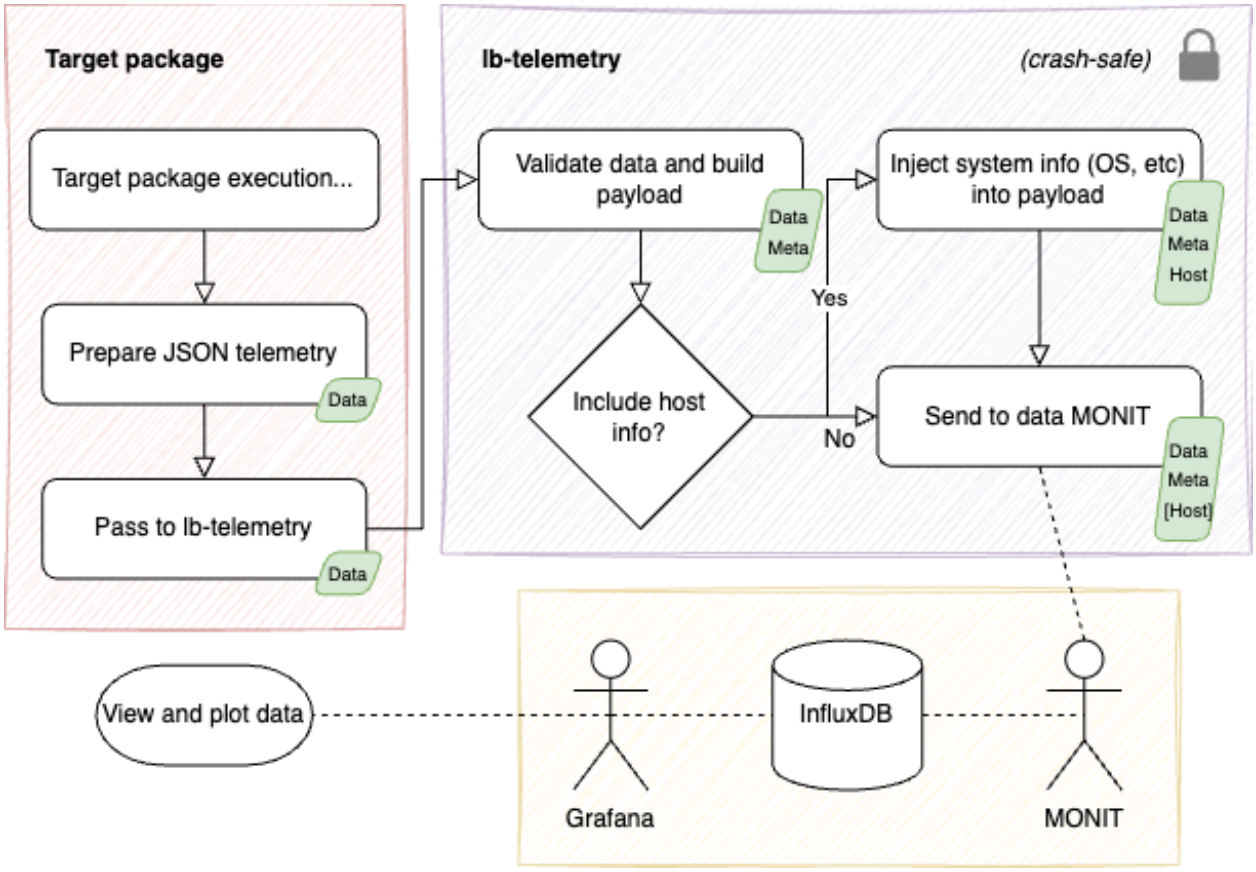

Figure 5: A diagram showing the flow of execution.

3.2 The CLI

One crucial part of `lb-telemetry` is its CLI. It enabled its addition to non-python systems (such as LbEnv, which is shell script), allowing telemetry to be sent from the command line, shell scripts, and more. The CLI also made it quick to test whether InfluxDB was receiving the data and generally made debugging easier.

3.3 Anonymity

It was important that the user was not forced into sending telemetry data while using a target package (for privacy/performance concerns or otherwise). We, therefore, added a variable `LBTELEMETRY_ENABLED=x` to `cern profile.sh` so that the user had to actively opt-in to send telemetry. We also made sure that no personally identifiable data is ever sent to ensure anonymity.

While setting up `lb-telemetry`, each target package may decide whether to send certain system information with the payloads via an `include_host_info` option. This includes data like the version of Python being used, the system architecture, and the operating system being used.

3.4 Error handling

It was important that the addition of `lb-telemetry` to any target package did not cause any fatal crash or otherwise interrupt the target's normal operation. It should sit in the background and be mostly invisible to the user while quietly collecting and logging telemetry. This requirement meant that error handling was carefully considered and implemented such that any errors raised by `lb-telemetry` were handled and did not crash the target package.

4 Logging and graphing

4.1 MONIT and InfluxDB

MONIT is a CERN data monitoring service that provides access to tools such as InfluxDB (database) and Grafana (graphing). It was an essential tool in the project as it saved us from setting up InfluxDB/Grafana instances manually and kept our telemetry close to the rest of the LHCb monitoring data. Sending data to MONIT involves a simple HTTP request, stating the data to be inserted and the values that are tags. More detailed info can be found on the docs.

```
payload = {
  "producer":  "lb-telemetry",
  "type":  "SomeTable",
  "timestamp":  1689940164,
  "idb_tags":  ["tag1"],
  "tag1":  "tag_value",
  "field1":  "field_value",
}
```

It is worth noting that InfluxDB tables have a (mostly) immutable schema. Measurement tags/fields should be firmly decided before adding `lb-telemetry` to a target package. After a measurement is added to a table, the table's tags/fields cannot be renamed or removed, although new ones can still be added. In the case that the schema must be changed, a new table should be created to start with a clean slate.

4.2 Grafana

When the data has arrived in InfluxDB (usually in under a minute), Grafana can directly access it. It is easy to add/edit plots as required to best represent and explore the data gathered by each target package. Grafana is intuitive to use and my previous experience with the

tool helped me set up the dashboards quickly. We initially had a single `lb-telemetry` dashboard with a row of plots for each target package but as more plots were added, it became clear that each target package deserved its own dashboard. They can be found here in the LbTelemetry folder.

A special `Tests` dashboard exists to monitor some functions of `lb-telemetry` itself. At the top of the dashboard, the timestamp of the most recent test measurement that was inserted into the database is displayed. This is useful for verifying that everything is working end-to-end when a new version of `lb-telemetry` is released. A table plot lists recent messages sent to the CLI table and a plot below shows the number of missed telemetry events collected by the `telemetry-proxy`.

5 Telemetry Proxy

5.1 Introduction and purpose

The `telemetry-proxy` project was an extension to `lb-telemetry` to allow telemetry to be gathered from package users who are outside the CERN network (since MONIT only accepts internal connections). When users disable telemetry collection, the proxy can give an idea of the number of missed telemetry events by incrementing a counter rather than sending the full details. The `telemetry-proxy` is an HTTP server built in Rust and runs in a Docker container on CERN's OpenShift instance.

5.2 Usage

The `telemetry-proxy` is never directly used. Instead, it is accessed indirectly and automatically when `lb-telemetry` is used outside the CERN network. Nevertheless, an example of its direct usage are shown below for interest and testing purposes. Full usage information can be found in the `README` of the `telemetry-proxy` project.

```
curl -vX POST https://lb-telemetry-proxy.app.cern.ch/disabled -d '"producer":
"lb-telemetry", "type":  "testing", "timestamp":  1692024685459, "test_field":
"test_value"' --header "Content-Type:  application/json"
```

5.3 Endpoints

`/disabled`

Used when the sending of full telemetry is disabled. Records the fact that some piece of telemetry destined for table `type` at time `timestamp` was missed.

`/proxy`

Used when sending full telemetry from outside the CERN network since MONIT drops

external connections. The body should be a valid JSON string that matches MONIT's spec. Only select, whitelisted `types` can be used.

5.4 Data validation and rate limiting

The proxy's endpoints will eventually be exposed to the public, so extensive data validation and rate limiting are put in place to block malicious requests and protect MONIT from being flooded with traffic. Requests must exactly match the expected format and be destined for a whitelisted table if being forwarded to MONIT. There is also rate limiting in place, allowing up to 60 requests per minute with a burst cap of 20 requests. The client's request allowance refreshes at 1 request per second up to this cap. The proxy will respond with status code 429 to requests sent after the rate limit has been exceeded.

6 Learning Achievements

Working on this project filled in numerous knowledge gaps that I didn't know I had. Despite my proficiency in Python, it was still novel and challenging working in a scientific context and in a large organisation where code quality, maintainability, and readability were paramount. The project also introduced me to thorough testing, continuous integration and deployment, and preparing/publishing a Python package. Having the chance to apply my background in InfluxDB and Grafana was satisfying and made me feel like a valuable asset to the team. I learned a lot about how a large organisation operates with respect to communication, toolchains, and contributing to other projects via forking/branching and merge requests. It has always been an ambition of mine to work in an interdisciplinary environment and apply my skills to real-world problems and I would certainly say I've achieved that. Having several opportunities to present my work to the broader team via posters/slideshows was extremely rewarding and memorable.