

Profiling Apache Beam Python pipelines

Israel Herraiz
Strategic Cloud Engineer,
Google Cloud
ihr@google.com
[@herraiz](https://twitter.com/herraiz) (Twitter)



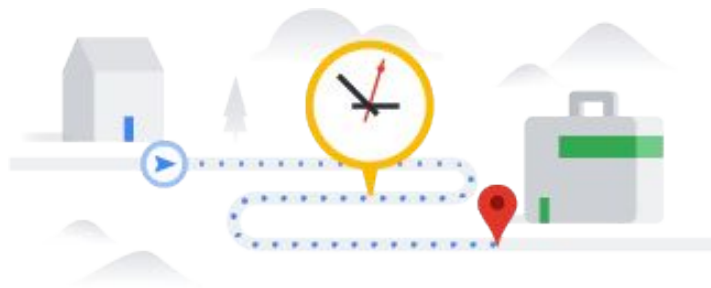
Have your customers ever asked you about profiling Python Dataflow pipelines?

And you answered it is not possible...

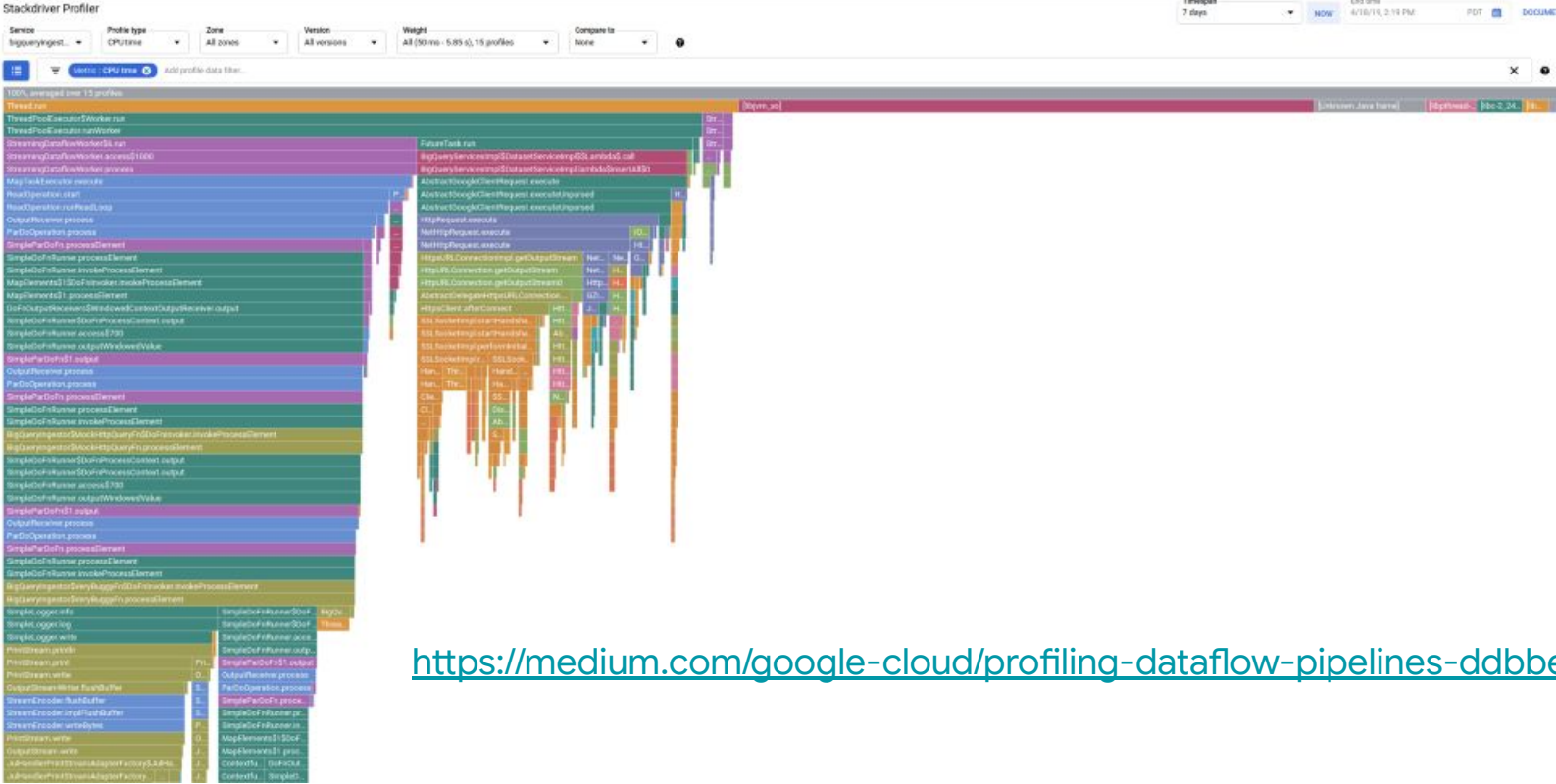
Profiling?

Finding **bottlenecks** in your pipeline

and **mapping** the bottlenecks to
specific locations in the **source code**



Neatly integrated in GCP for Java Dataflow pipelines

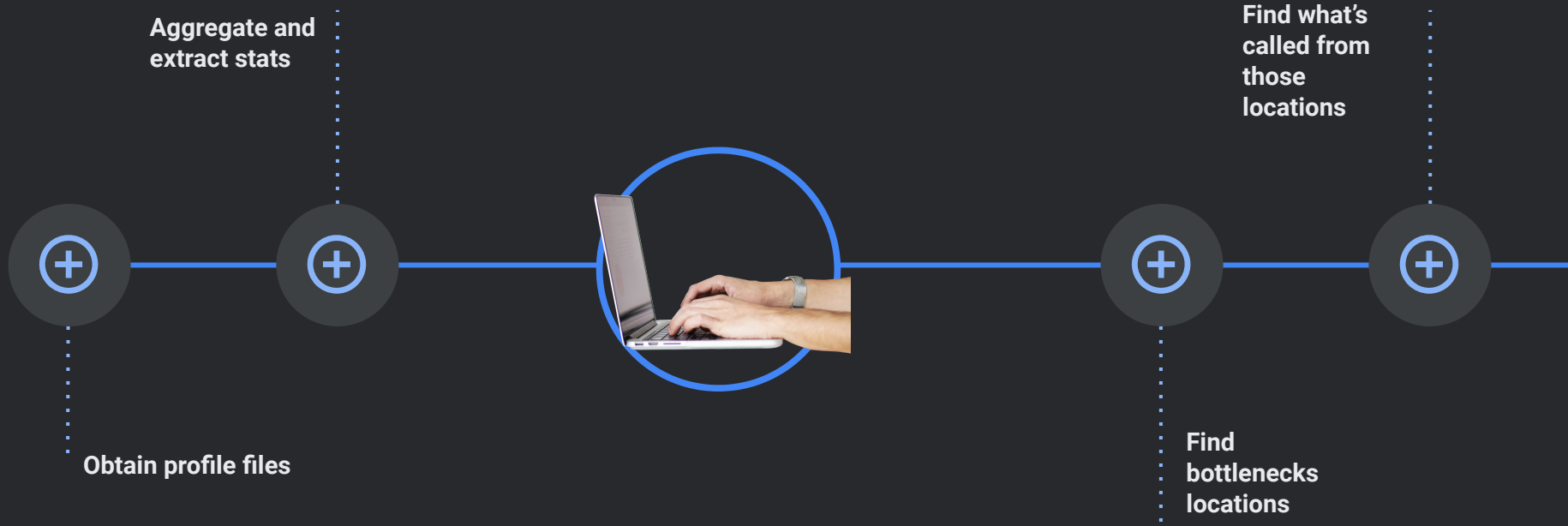


<https://medium.com/google-cloud/profiling-dataflow-pipelines-dbbef07761d>

<https://github.com/iht/python-profiling-beam-summit-2021>

Not so much for Python... (*)

Cloud Human Profiler



Profiling options in Apache Beam

profile_cpu

profile_memory

profile_location

profile_sample_rate














```
913 class ProfilingOptions(PipelineOptions):
914     @classmethod
915     def _add_argparse_args(cls, parser):
916         parser.add_argument(
917             '--profile_cpu',
918             action='store_true',
919             help='Enable work item CPU profiling.')
920         parser.add_argument(
921             '--profile_memory',
922             action='store_true',
923             help='Enable work item heap profiling.')
924         parser.add_argument(
925             '--profile_location',
926             default=None,
927             help='path for saving profiler data.')
928         parser.add_argument(
929             '--profile_sample_rate',
930             type=float,
931             default=1.0,
932             help='A number between 0 and 1 indicating the ratio '
933             'of bundles that should be profiled.')
934
```

[Link to Apache Beam ProfilingOptions](https://github.com/iht/python-profiling-beam-summit-2021)

Launch your pipeline and add the CPU profiling options

```
python main.py --project=$PROJECT_ID \  
  --runner=DataflowRunner \  
  --region=$REGION \  
  --streaming \  
  --use_public_ips \  
  --requirements_file ./requirements.txt \  
  --setup_file ./setup.py \  
  --temp_location=$TMP_LOCATION \  
  --profile_cpu \  
  --profile_location=$PROFILE_LOCATION \  
  --input-topic=$INPUT_TOPIC \  
  --output-table-rides=$OUTPUT_TABLE RIDES \  
  --output-table-aggregate=$OUTPUT_TABLE_AGG
```

Dataflow will upload a profile file **per bundle**

<input type="checkbox"/>	Name	Size
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-12	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-14	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-16	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-18	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-2	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-20	33.4 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-24	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-26	33.9 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-32	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-36	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-4	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-6	34.1 KB
<input type="checkbox"/>	 2021-05-08_22_07_18-process_bundle-8	34.1 KB

Obtain profile files

What are these errors?

Enabling profiling will cause lots of errors to be reported in Dataflow

The screenshot shows the 'EXECUTION DETAILS' tab for a Dataflow job. At the top right, a red circle highlights a notification icon with the number '12344'. Below this, the 'Logs' section is active, showing a list of error messages. A red arrow points from the error count icon to the 'Error' severity filter. The error messages are all 'Traceback' errors from the 'generic:unknown' package, indicating a failure to read from a PubSub topic. The messages are as follows:

- Scanned up to 5/9/21, 12:36 AM. Scanned 12.1 MB.
- 2021-05-09 00:36:23.836 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:24.199 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:24.566 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:24.792 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:25.063 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:25.089 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:25.117 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:25.155 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker
- 2021-05-09 00:36:25.262 CEST Error message from worker: generic:unknown: Traceback (most recent call last): File "/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker

Python profiler

The files are produced by the Python profiler
<https://docs.python.org/3/library/profile.html>

Custom prof files, not used by any other profiling tools

You need to use pstats from Python, to aggregate your profiling data

```
import os
import pstats

directory = 'data'

files = ['%s/%s' % (directory, x) \
         for x in os.listdir('%s/' % directory)]
p = pstats.Stats(*files)
p.sort_stats('cumulative').dump_stats('dataflow.prof')
```

150 MB of bundle files
with **1** as bundle ratio
are converted into
a single file of **117 KB**

Aggregated metrics explained

Number of times a function is called

```
p = pstats.Stats('dataflow.prof')
p.sort_stats('cumulative').print_stats()
```

Code location (file and line)

CPU time

- Total time
- Cumulative

What's the difference?

```
Sat Nov 21 22:06:08 2020      dataflow.prof

      903110366 function calls (855530510 primitive calls) in 10429.669 seconds

Ordered by: Total CPU time
# times called
ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
42783   21.000   0.000  9470.117   0.221  /usr/local/lib/python3.8/site-packages/apache_beam/runners/worker/bundle_processor.py:920(process_bundle)
2964    4.986   0.002  5065.839   1.709  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery.py:1302(_flush_batch)
2964    5.880   0.002  5057.478   1.706  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery_tools.py:943(insert_rows)
441998  0.968   0.000  4864.982   0.011  apache_beam/runners/common.py:1243(_invoke_bundle_method)
441998/220999  1.314   0.000  4822.989   0.022  apache_beam/runners/worker/operations.py:730(finish)
220999  0.461   0.000  4821.187   0.022  apache_beam/runners/common.py:1265(finish)
441998/220999  1.022   0.000  4820.166   0.022  apache_beam/runners/common.py:514(invoke_finish_bundle)
2938    0.273   0.000  4815.455   1.639  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery.py:1273(finish_bundle)
2938    0.120   0.000  4814.822   1.639  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery.py:1290(_flush_all_batches)
2938    0.862   0.000  4814.417   1.639  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery.py:1296(<listcomp>)
2964    0.431   0.000  4440.970   1.498  /usr/local/lib/python3.8/site-packages/apache_beam/utils/retry.py:225(wrapper)
2964    0.381   0.000  4440.390   1.498  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/bigquery_tools.py:511(_insert_all_rows)
2964    0.124   0.000  4435.281   1.496  /usr/local/lib/python3.8/site-packages/apache_beam/io/gcp/internal/clients/bigquery/bigquery_client.py:1296(_insert_rows)
2964    0.442   0.000  4433.805   1.496  /usr/local/lib/python3.8/site-packages/apitools/base/py/base_api.py:689(_RunMethod)
2964    0.470   0.000  3874.896   1.307  /usr/local/lib/python3.8/site-packages/apitools/base/py/base_api.py:658(PrepareHttpRequest)
5928    0.232   0.000  3860.671   0.651  /usr/local/lib/python3.8/site-packages/apitools/base/py/encoding_helper.py:115(MessageToJson)
```

CPU metrics



cumtime

Wall clock time

This includes the time spent in your code, plus any other time spent elsewhere.

This includes:

- Time waiting for another function
- Time waiting for an API request to be finished
- Other sources of slowness (I/O)

In most occasions, this will be the metric you want to use to profile your pipeline.

Lots of performance issues are due to:

- I/O
- Slow external APIs
- Third party libraries called from your code



tottime

Busy CPU time

This is strictly the CPU time used to execute the code inside your function.

It does not count any time spent inside other functions called from your code.

This metric is relevant if you are implementing algorithms, and are interested in performance bottlenecks related to algorithmic complexity.

Soooo much calls from the Beam SDK.

Dude, where is my code?

Sort prof dictionary by value and filter for custom code

```
def is_my_key(key, modules):
    fn = key[0]
    for m in modules:
        module_name = '%s' % m
        if module_name in fn:
            return True
    return False
```

```
stats_dict = p.sort_stats('cumulative').stats
```

```
mymodules = ['dofns', 'pipeline']
mykeys = [k for k in stats_dict.keys() if is_my_key(k, mymodules)]
mystats = {k: stats_dict[k] for k in mykeys}
```

```
percall = {}
```

```
for k,v in mystats.items():
    _, ncalls, _, cumtime, _ = v
    avg_time = cumtime/ncalls
    percall[k] = avg_time
```

```
sorted(percall.items(), key=lambda x: x[1], reverse=True)
```

We sort the dictionary of stats per value (using `cumtime`).

However, we have **limited observability**

- We only get metrics at method/function level

Importance of splitting **large process** methods into **several functions**.

```
[('/usr/local/lib/python3.8/site-packages/dofns/business_rules.py',
 22,
 'process'),
 0.002303815858387165),
 ('/usr/local/lib/python3.8/site-packages/dofns/business_rules.py',
 80,
 '_parse_timestamp'),
 0.0013371157256475503),
 ('/usr/local/lib/python3.8/site-packages/dofns/parse_messages.py',
 31,
 'process'),
 0.00023117240416898547),
```

process method of DoFn

Average time

A function called inside that process method

Average time

We have found the slowest methods in our pipeline.

If you don't write small functions, most likely you will find a process method of a DoFn.

This is the same info you get in the Dataflow job page.

So please write small functions!

What is being called from my slowest functions?

```
def find_external_functions(key, stats):
    output = {}
    for k,v in stats.items():
        _, _, _, _, d = v
        if key in d:
            output[k] = v

    return output
```

```
# Let's assume that we want to find all the code called from mykeys[1]
external_stats = find_external_functions(mykeys[1], stats_dict)
percall = {}
```

```
for k,v in external_stats.items():
    _, ncalls, _, cumtime, _ = v
    avg_time = cumtime/ncalls
    percall[k] = avg_time
```

```
sorted(percall.items(), key=lambda x: x[1], reverse=True)
```

Finding the slowest functions is normally not enough.

Why are those functions so slow?

- In this code snippet, we find all the functions that are called from the location with index 1 (second slowest).
- The first (index 0) was the **process** method of a DoFn.
 - Func with index 1 is called from that method

96% of the avg. time is due to the calls to the **python-dateutil** package

- That time is approx. 50% of the **process** method time
- This is an open source package for handling dates with Python

```
[({'/usr/local/lib/python3.8/site-packages/dateutil/parser/_parser.py',
  1276,
  'parse'},
  0.001292679971553465),
  {'/usr/local/lib/python3.8/site-packages/apache_beam/runners/worker/statesampler.py',
  54,
  'get_current_tracker'},
  9.479584844738897e-06),
  {'apache_beam/runners/worker/statesampler_fast.pyx', 196, 'update_metric'},
  2.908057548493207e-06]
```

← Average time

← Package name

**So can you profile
Apache Beam and
Dataflow Python
pipelines?**

Now you can!

DIY

Repository with sample streaming pipeline in Python

<https://github.com/iht/python-profiling-beam-summit-2021>

Analysis of prof file:

https://colab.research.google.com/drive/1fmefgXctJWxyVv0_CXsQ9Hyfep488yfN

Get started with GCP:

<https://cloud.google.com/free/>

Take away

Profiling Dataflow pipelines

The most common performance problems are related to the implementation of the pipeline.

Is Python supported?

Not as neat as Java with Dataflow and GCP, but you can obtain valuable profiling data with a little bit of scripting.

WSF

Write

Small

Functions