Profiling Apache Beam Python pipelines

Israel Herraiz
Strategic Cloud Engineer, Google Cloud
ihr@google.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-ddbbef07761d

https://github.com/iht/python-profiling-beam-summit-2021
Not so much for Python… (*)

https://github.com/iht/python-profiling-beam-summit-2021
Cloud Human Profiler

- Obtain profile files
- Aggregate and extract stats
- Find bottlenecks locations
- Find what's called from those locations

https://github.com/iht/python-profiling-beam-summit-2021
Profiling options in Apache Beam

- `profile_cpu`
- `profile_memory`
- `profile_location`
- `profile_sample_rate`

---

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

Link to Apache Beam ProfilingOptions
Launch your pipeline and add the CPU profiling options

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

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

https://github.com/iht/python-profiling-beam-summit-2021
What are these errors?

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

https://github.com/iht/python-profiling-beam-summit-2021
Python profiler

The files are produced by the Python profiler [https://docs.python.org/3/library/profile.html](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

```python
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**
- **Code location (file and line)**
- **CPU time**
  - Total time
  - Cumulative

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

```python
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.
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?

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

```python
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)
```

https://github.com/iht/python-profiling-beam-summit-2021
So can you profile Apache Beam and Dataflow Python pipelines?

Now you can!

https://github.com/iht/python-profiling-beam-summit-2021
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/1fmefqXctJWxyVv0_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