Troubleshooting Python pipelines with process monitoring tools

Valentyn Tymofieiev

https://s.apache.org/how-to-spy-on-python-sdk-harness

BEAM SUMMIT

September 4-5, 2024

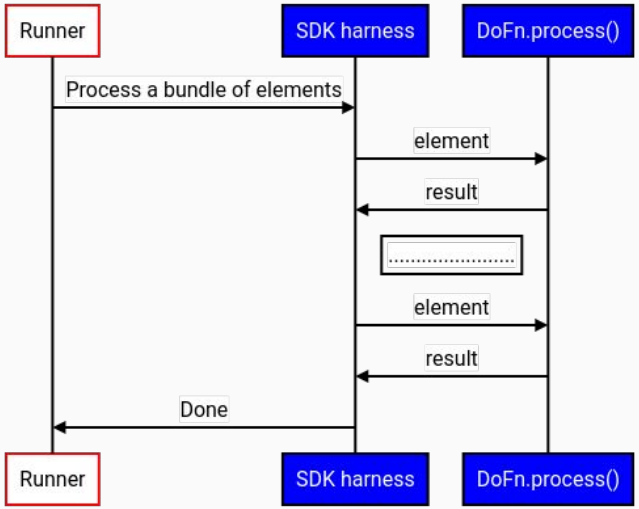
Sunnyvale, CA. USA

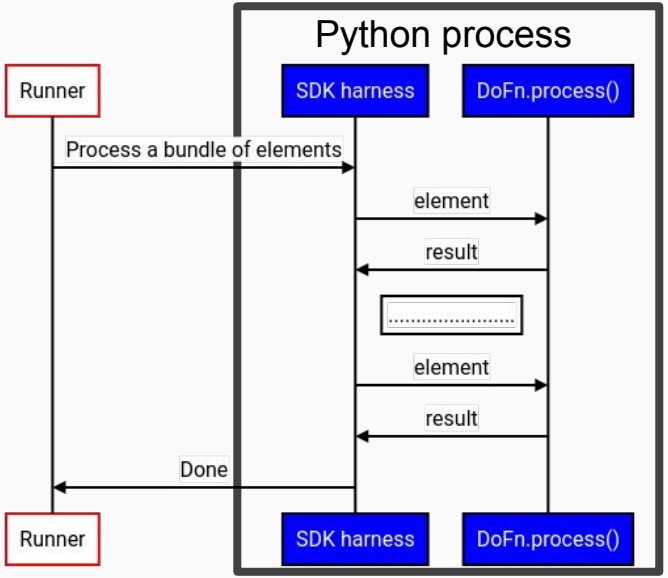
How a pipeline might fail

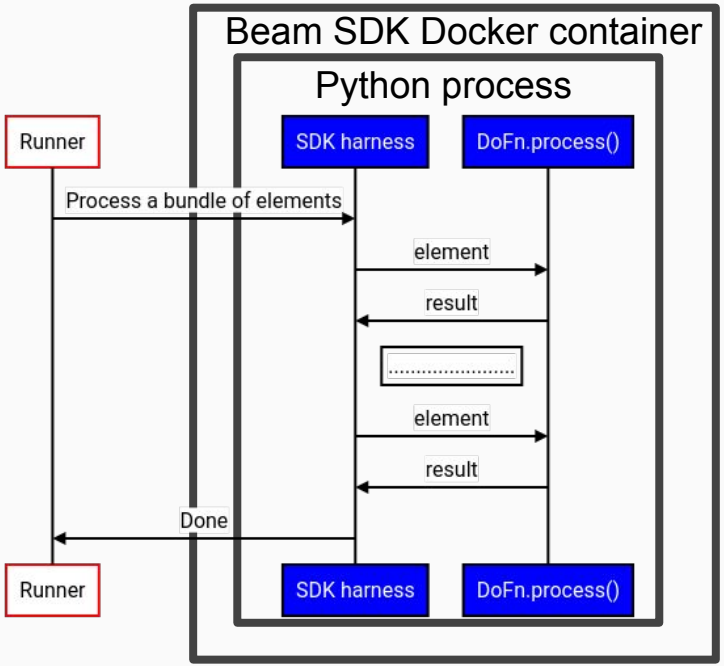
- Ideal case: Pipeline doesn't fail
- Not ideal case: Pipeline fails, but produces a clear and actionable error
- Unfortunate case: Pipeline fails, and there is no clear or actionable error

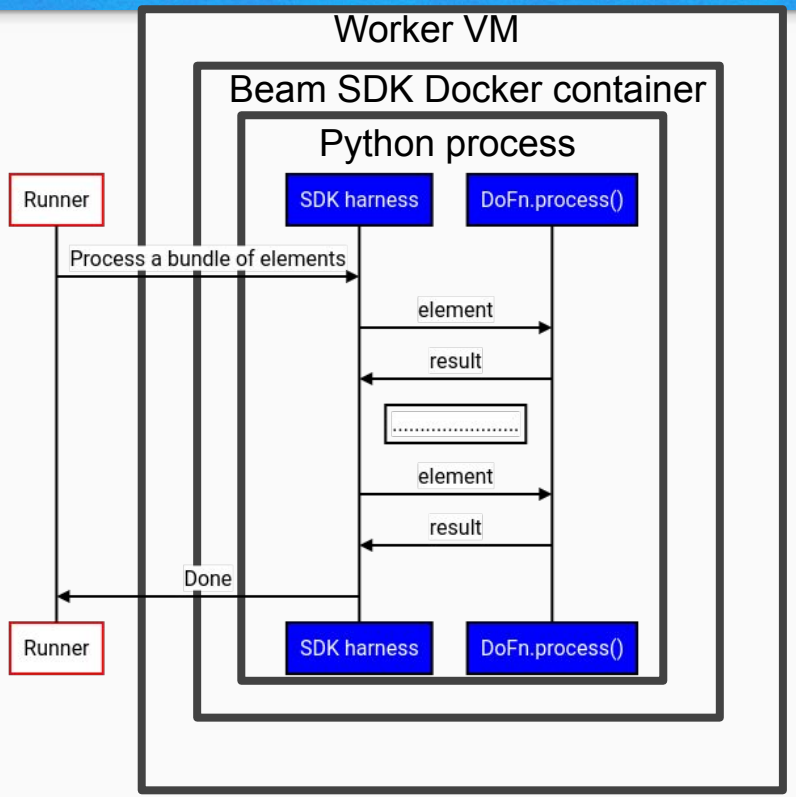
How a pipeline might fail

- Ideal case: Pipeline doesn't fail
- Not ideal case: Pipeline fails, but produces a clear and actionable error
- **Unfortunate case: Pipeline fails, and there is no clear or actionable error**


Runner
SDK harness
DoFn.process()
Process a bundle of elements
element
result
.....
element
result
Done
Runner
SDK harness
DoFn.process()


Python process
Runner
SDK harness
DoFn.process()
Process a bundle of elements
element
result
.....
element
result
Done
Runner
SDK harness
DoFn.process()


Beam SDK Docker container
Python process
Runner
SDK harness
DoFn.process()
Process a bundle of elements
element
result
.....
element
result
Done
Runner
SDK harness
DoFn.process()

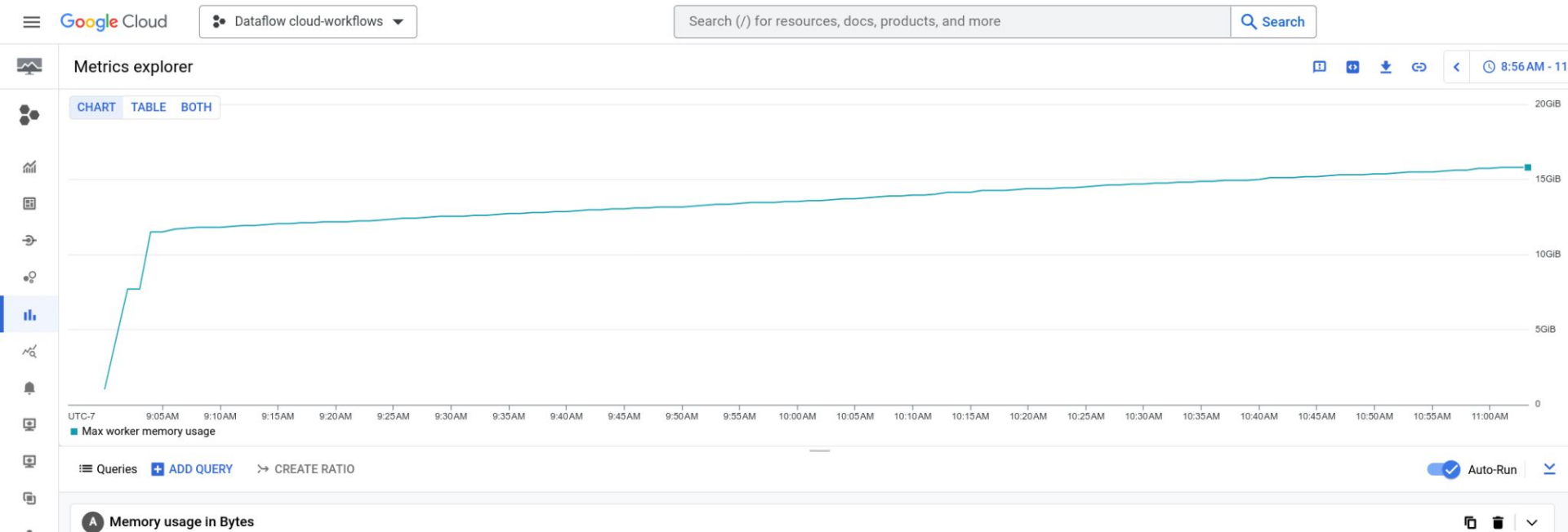

Worker VM
Beam SDK Docker container
Python process
Runner
SDK harness
DoFn.process()
Process a bundle of elements
element
result
.....
element
result
Done
Runner
SDK harness
DoFn.process()

What clues can we find from observing SDK harness process?

Case 1: Identifying a Memory Leak

Symptoms

Worker RAM usage in a pipeline increases overtime.



Instrument SDK harness process with a profiler

Consideration for choosing a profiler

- Detects the leak
- Doesn’t require the process to finish
- Language-aware
- Easy to instrument and use
- Still maintained

Instrument SDK harness process with a profiler

Consideration for choosing a profiler

- Detects the leak
- Doesn’t require the process to finish
- Language-aware
- Easy to instrument and use
- Still maintained

Memray: https://bloomberg.github.io/memray/

Instrument Beam SDK container with a profiler

Container image definition: sdks/python/container/Dockerfile:

```
FROM <Linux base image with Python>
RUN pip install <Beam dependencies>
ENTRYPOINT = <A go binary that launches the SDK>
```

Entrypoint definition: sdks/python/container/boot.go

// <Download pipeline dependencies>

// <Create a venv and install runtime packages>

// <Launch SDK harness>:

python -m apache_beam.runners.worker.sdk_worker_main <args>

Instrument Beam SDK container with a profiler

Container image definition:

```
FROM <Linux base image with Python>
RUN pip install <Beam dependencies> memray
ENTRYPOINT = <A go binary that launches the SDK>
```

Instrumented entrypoint definition

```
// <Download pipeline dependencies>

// <Create a venv and install runtime packages>

// <Launch SDK harness>:

   memray run python -m apache_beam.runners.worker.sdk_worker_main <args>
```

Instrument Beam SDK container with a profiler

Container image definition:

```
FROM <Linux base image with Python>
RUN pip install <Beam dependencies> memray
ENTRYPOINT = <A go binary that launches the SDK>
```

Instrumented entrypoint definition

```
// <Download pipeline dependencies>

// <Create a venv and install runtime packages>

// <Launch SDK harness>:

   memray run python -m apache_beam.runners.worker.sdk_worker_main <args>
```

Build command: gradlew :sdks:python:container:py310:docker

Creates a local image: apache/beam_python3.10_sdk:2.59.0

Much easier if you can repro a leak locally

beam_pipeline.py:

```
import argparse
import apache_beam as beam

parser = argparse.ArgumentParser()
_, pipeline_args = parser.parse_known_args()

with beam.Pipeline(argv=pipeline_args) as p:
  p | beam.Create([1]) | beam.Map(lambda x: x+1)
```

:$ pip install apache-beam==2.47.0

:$ pip install memray

:$ memray run -o output.bin --force beam_pipeline.py --runner Direct --direct_runner_bundle_repeat=10000

memray flamegraph report

:$ memray flamegraph --leak --force output.bin

memray table report

:$ memray table --leak --force output.bin

memray table report (memory leaks)

Python Allocator: pymalloc | Stats | Help | Search

Report generated using "--leaks" using pymalloc allocator

This report for memory leaks was generated with the pymalloc allocator active, but without tracking enabled for calls to the pymalloc allocator. This will show confusing results because the pymalloc allocator will retain memory in memory pools even after the objects that requested that memory are deallocated, and Memray won't be able distinguish memory set aside for reuse from leaked memory. You should rerun your application with the `PYTHONMALLOC=malloc` environment variable set or pass the `--trace-python-allocators` flag when profiling your application. Click here for more information.

Thread ID	Size	Allocator	Allocations	Location
0x1	4.0 MiB	realloc	6	monitoring_infos at /home/valentyn/.pyenv/versions/3.10.13/envs/py310/lib/python3.10/site-packages/apache_beam/runners/worker/bundle_processor.py:1195
0x1	2.6 MiB	malloc	109	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.3 MiB	malloc	249	_call_with_frames_removed at <frozen importlib._bootstrap>:241
0x1	1.1 MiB	calloc	193	_call_with_frames_removed at <frozen importlib._bootstrap>:241
0x1	1.1 MiB	malloc	22	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.1 MiB	malloc	66	BuildMessage at /home/valentyn/.pyenv/versions/3.10.13/envs/py310/lib/python3.10/site-packages/google/protobuf/internal/builder.py:85
0x1	1.1 MiB	malloc	29	__subclasscheck__ at /home/valentyn/.pyenv/versions/3.10.13/lib/python3.10/abc.py:123
0x1	1.0 MiB	malloc	34	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.0 MiB	malloc	23	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.0 MiB	malloc	21	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.0 MiB	malloc	21	_call_with_frames_removed at <frozen importlib._bootstrap>:241

Thread ID	Size	Allocator	Allocations	Location
0x1	4.0 MiB	realloc	6	monitoring_infos at /home/.../apache_beam/runners/worker/bundle_processor.py:1195
0x1	2.6 MiB	malloc	109	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.3 MiB	malloc	249	_call_with_frames_removed at <frozen importlib._bootstrap>:241
0x1	1.1 MiB	calloc	193	_call_with_frames_removed at <frozen importlib._bootstrap>:241
0x1	1.1 MiB	malloc	22	_compile_bytecode at <frozen importlib._bootstrap_external>:672
0x1	1.1 MiB	malloc	66	BuildMessage at /home/valentyn/.pyenv/versions/3.10.13/envs/py310/lib/python3.10/site-p
0x1	1.1 MiB	malloc	29	__subclasscheck__ at /home/valentyn/.pyenv/versions/3.10.13/lib/python3.10/abc.py:123

Use *memray run --native* to trace (C/C++) stack frames

Thread ID	Size	Allocator	Allocations	Location
0x1	32.0 MiB	realloc	6	_upb_Arena_SlowMalloc at <unknown>:0
0x1	4.0 MiB	realloc	6	_upb_Arena_SlowMalloc at <unknown>:0
0x1	2.5 MiB	malloc	1	_PyObject_Malloc at Objects/obmalloc.c:1966
0x1	1.0 MiB	mmap	1	_PyObject_ArenaMmap at Objects/obmalloc.c:150
0x1	1.0 MiB	mmap	1	_PyObject_ArenaMmap at Objects/obmalloc.c:150
0x1	1.0 MiB	mmap	1	_PyObject_ArenaMmap at Objects/obmalloc.c:150
0x1	1.0	mmap	1	_PyObject_ArenaMmap at Objects/obmalloc.c:150

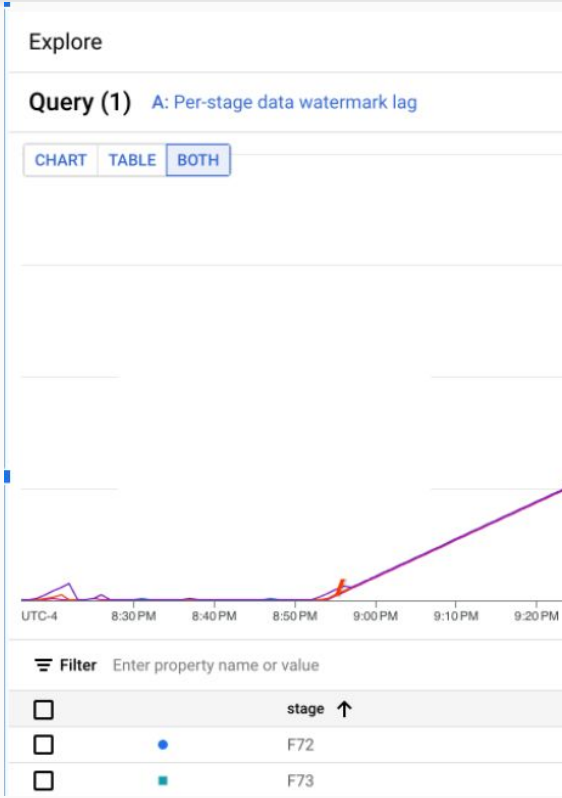
Case 2: Root-causing stuckness: What holds the GIL?

Symptoms

- Pipeline is stuck
- Dataflow Runner cannot communicate with Beam SDK:

  Unable to retrieve status info from SDK harness

  There are 10 consecutive failures obtaining SDK worker status info. SDK worker appears to be permanently unresponsive. Aborting the SDK.



What exactly is the "status info" ?

- BeamFnWorkerStatus: API for SDKs to report status to a runner.
- Part of Beam Fn API: https://s.apache.org/beam-fn-api-harness-status

Sample (available on Dataflow workers at: localhost:8081/sdk_status):

```
========== ACTIVE PROCESSING BUNDLES ==========
--- instruction process_bundle-4485482240419851547-7244 ---
ProcessBundleDescriptorId: s24-111
tracked thread: <_Worker(Thread-26, started daemon 140013912979200)>
time since transition: 1.13 seconds
...

========== THREAD DUMP ==========
--- Thread #140026413855936 name: MainThread ---
  File "/usr/local/lib/python3.9/runpy.py", line 197, in _run_module_as_main
    return _run_code(code, main_globals, None,

...
```

Finding why SDK refuses to give status info

- SDK harness serves status in a background thread
- Hypothesis: Could it be that some other thread holds the GIL indefinitely?

  Let's find out!

Finding why SDK refuses to give status info

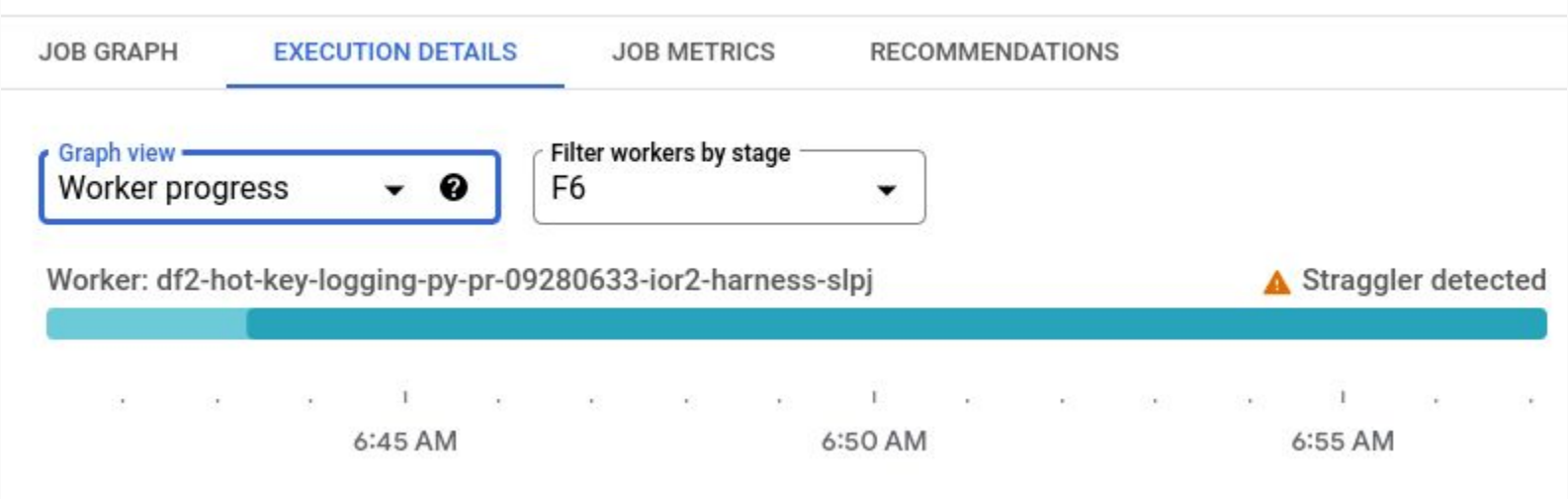
1) Repro the issue in a running pipeline

2) Find a worker that is stuck

3) Locate the Python SDK harness process on the worker

4) Inspect the process stack frames for running threads

Find a worker that is stuck



Locate the running SDK process

SSH into the worker:

gcloud compute **ssh** --zone "us-central1-b" "beamapp-someworker-harness-abcd" --project "project-id"

Verify that SDK worker is not responsive.

:$ curl localhost:8081/sdk_status # <no output>

Find a container running the Python SDK harness.

:$ docker ps
look for beam sdk container that has 'python' in its name, note its hash, then log into the running container:

CONTAINER ID IMAGE
6577f349f06d ...beam_python3.8_sdk
...

Get a shell with a privileged mode inside the running container:

CONT_ID=`docker ps | grep python | awk '{print $1}'` ; docker exec **--privileged** -it $CONT_ID /bin/bash

Inspect the SDK process

Inside SDK harness container in privileged mode

```
:$ ps -A

    PID TTY          TIME CMD
      1 ?        00:00:00 boot         <------  SDK container entrypoint
     29 ?        00:03:02 python       <------  SDK harness python process
     30 ?        00:02:46 python       <------  SDK harness python process
...
```

:$ pip install **pystack**

:$ pystack remote 29

Find which thread holds the GIL

```
root@beamapp-valentyn-04101932-04101232-3i3x-harness-3sqg:/# pystack remote 29

Traceback for thread 100 (python) [Has the GIL] (most recent call last):
    (Python) File "/usr/local/lib/python3.8/threading.py", line 890, in _bootstrap
        self._bootstrap_inner()
    ...
    (Python) File "/usr/local/lib/python3.8/site-packages/google/cloud/bigtable/client.py", line
285, in _create_gapic_client_channel
        channel = grpc_transport.create_channel(
    ...
    (Python) File "/usr/local/lib/python3.8/site-packages/grpc/_channel.py", line 2046, in __init__
        self._channel = cygrpc.Channel(

Traceback for thread 99 (python) [] (most recent call last):
...
```

Inspect native (C/C++) frames if necessary

pystack:

```
# To look up the native, use --native-all flag or use gdb:
pystack remote --native-all $PYTHON_PID
```

gdb:

```
apt update && apt install -y gdb
gdb --quiet \
  --eval-command="set pagination off" \
  --eval-command="thread apply all bt" \
  --eval-command "set confirm off" \
  --eval-command="quit"  -p $PYTHON_PID
```

BEAM SUMMIT

Case 3: Segmentation fault

Symptoms

Pipeline crashed with an error: *segmentation fault (core dumped)*

>	2023-04-12 18:41:05.788 PDT	2023/04/13 01:41:05 Python (worker sdk-0-0) exited 2 times: signal: segmentation fault (core dumped)
>	2023-04-12 18:41:05.788 PDT	restarting SDK process
>	2023-04-12 18:41:05.789 PDT	2023/04/13 01:41:05 Executing Python (worker sdk-0-0): python -m apache_beam.runners.worker.sdk_worker_main
>	2023-04-12 18:41:05.789 PDT	Completing WorkerStatus() connection for SDK harness sdk-0-0 which is unexpected unless the job is being terminated.
>	2023-04-12 18:41:05.789 PDT	SDK harness sdk-0-0 disconnected.

... core dumped where? can we actually access the core file?

Collecting core files with a custom container

Dockerfile:

```
FROM apache/beam_python3.9_sdk:2.58.0

# Use a modified entrypoint

COPY ./entrypoint_that_uploads_core_files.sh /opt
RUN chmod +x /opt/entrypoint_that_uploads_core_files.sh
ENTRYPOINT ["/opt/entrypoint_that_uploads_core_files.sh"]
```

Run the pipeline:

```
python pipeline.py \
  --runner=Dataflow \
  --experiments "core_pattern=/core_%t.%h.%e.%p" \
  --sdk_container_image=<your image>
  ...
```

entrypoint_that_uploads_core_files.sh:

```
#!/bin/bash
upload_core_files_periodically() {
  while true; do
    gsutil -m cp -n  /*core* gs://my_bucket/core_files/ || true
    sleep 1
  done
}

upload_core_files_periodically &

# ..but also launch regular Beam entrypoint
/opt/apache/beam/boot "$@"

# and just in case another upload when entrypoint exits
gsutil -m cp -n  /*core* gs://my_bucket/core_files || true
```

Analyzing core files

Replicate a runtime environment from container image.

```
mkdir /tmp/core_files
gsutil cp -r gs://my_bucket/core_files/  /tmp/core_files

docker run --rm -it \
  --entrypoint=/bin/bash \
  -v /tmp/core_files:/tmp/core_files \
  <your_container_image>
```

Analyze the core file with pystack or GDB:

<install pystack or gdb>

```
:$ pystack core /tmp/core_files/core_1725438012.python.575 /usr/local/bin/python

:$ gdb /usr/local/bin/python /tmp/core_files/core_1725438012.python.575
```

Links for more information

- https://beam.apache.org/documentation/runtime/environments/
- https://cloud.google.com/dataflow/docs/guides/build-container-image
- https://cwiki.apache.org/confluence/display/BEAM/Investigating+Memory+Leaks
- https://cloud.google.com/dataflow/docs/guides/common-errors#worker-lost-contact

- Memray: The endgame Python memory profiler: https://bloomberg.github.io/memray/
- Pystack: The endgame Python stack debugger: https://bloomberg.github.io/pystack/

These slides: https://s.apache.org/how-to-spy-on-python-sdk-harness

Next steps

Thank you!

Questions?

Slides:

contact:

Valentyn Tymofieiev

tvalentyn@apache.org

GH: @tvalentyn