# Profiling Airflow tasks with Memray

Cedrik Neumann

## About me

- Mathematics and computer science in Berlin
- Data Engineer since 2013
- Working with Airflow since 2015
- First commit to Airflow in 2019
- Helped migrating King's data orchestration from Jenkins to Airflow on Astro Cloud from 2018 to 2024

## The problem

- Intermittent task failures + high memory usage
- Unstable/slow DAG parsing performance
- Not an expert on profiling Python code

## Meet Memray

“The endgame Python memory profiler

Memray tracks and reports memory allocations, both in Python code and in compiled extension modules.”

– *Memray*

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

## Hackday time! 🤓

## How? 🤔

## Leverage existing Airflow functionality

1. **Cluster Policies**: Monkey patch existing tasks
2. **Object Storage**: File interface for remote storage backends from providers
3. **Operator Extra Links**: Link to make reports directly available from the UI
4. **Flask Blueprints**: Serve reports via Airflow's web server

## Examples 👨🏼‍💻

## Monkey patch task

- A task policy is responsible for monkey patching selected tasks
- The task's execute method is wrapped in a function, which takes care of executing the task in the context of Memray's tracker, generating reports and copying files to object storage
- Cluster policies are available since Airflow 2.6

```python
@hookimpl
def task_policy(task: BaseOperator) -> None:
    if not is_run_memray(task):
        return

    task.execute = memray_func(task.execute)


def memray_func(f: C) -> C:

    @functools.wraps(f)
    def memray_execute(*args, **kwargs) -> Any:
        # pre task
        try:
            with memray.Tracker(destination):
                return f(*args, **kwargs)
        finally:
            # post task

    return memray_execute
```

## Upload results to object storage

- A local temporary directory is used for the profile and generated reports
- The object storage interface allows us to be agnostic about the destination object storage when copying all results
- Object storage is available since Airflow 2.8 and currently experimental ⚠️

```python
# pre task
tmp = TemporaryDirectory("memray")
folder = Path(tmp.name)
destination = memray.FileDestination(
  folder / "profile.bin",
)

# execute task ...

# post task
make_reports(folder)

# can be a remote instance of object storage
dst_folder = get_object_storage_path(context["ti"].key)

# copy all local results to dst folder
for file in ObjectStoragePath(folder).iterdir():
    file.copy(dst_folder / file.name)
```

## Link to results with operator extra link

- We check existence of report via object storage API and return the corresponding URL if it exists
- We can define extra links for all operators with **global_operator_extra_links**
- Global operator extra links are available since Airflow 1.10.4

```python
class MemrayStatsLink(BaseOperatorLink):
    name = "Memray Stats"

    def get_link(self, operator: BaseOperator,
      *, ti_key: TaskInstanceKey) -> str:

        folder = get_object_storage_path(ti_key)
        file = folder / "stats.json"

        if not file.exists():
            return ""

        return get_url(file, ti_key)


class MemrayPlugin(AirflowPlugin):
    name = "memray_plugin"

    global_operator_extra_links = [
        MemrayStatsLink(),
    ]
```

## Serve results via Flask blueprints

- Airflow's plugin mechanism allows to add endpoints for our reports to the web server with Flask blueprints
- The object storage API lets us create an open file handle, which can be served directly
- Flask blueprints are going to disappear in Airflow 3 (AIP-79) ⚠️

```
blueprint = Blueprint(
    name="memray",
    import_name=__name__,
    url_prefix="/memray",
)


@blueprint.get("/stats.json")
def stats():
    folder = get_object_storage_path(ti_key)
    file = folder / "stats.json"

    f = file.open("rb")
    return send_file(f, mimetype="application/json")


class MemrayPlugin(AirflowPlugin):
    name = "memray_plugin"

    flask_blueprints = [blueprint]
```

## Demo time 👀

# All good? 🧐

## Problems & limitations

- Control which tasks to profile (configuration/code vs. on-demand)
- Expose metrics/results in Airflow UI (maybe AIP-68*?)
- Show extra links only for relevant tasks
- Doesn't work with bash/k8s operator (new processes)
- Doesn't work with deferrable tasks (triggers)

**AIP-68 Extended Plugin Interface for React Views*

## Ideas / Nice to have

- Profile DAGs inside the DAG processor
- Profile deferrable tasks (triggers)
- Perform other ways of profiling (i.e. CPU)
- Airflow interface to run task in custom context
- Task flow decorator
- Profile entire Airflow processes

## Are we supposed to run this in production permanently?

- No, probably not
- Profiling can affect performance significantly
- This project acts as a POC on how to profile Airflow tasks remotely
- In production you likely want to profile tasks more selectively
- … or implement dedicated test DAGs to track memory usage over time

# Thank you! 🙏

Mexico City, Mexico

github.com/m1racoli

linkedin.com/in/cedrikneumann

github.com/m1racoli/airflow-memray