



Auto Program Parallelization across Compute Pools

# The Program Parallelization Problem

Parallelizing workloads to the cloud is a **complex, time-consuming** process that requires specialized **expertise**.

**Needs knowledge** of parallel algorithms, orchestration, images, containers, networking, load balancing, etc.

# Steps to Parallelize a Program

## **Break a program into tasks:**

Data parallelization: Function level (e.g. MapReduce).

Program level parallelization: Parallelize the call-graph (DAG Structure)



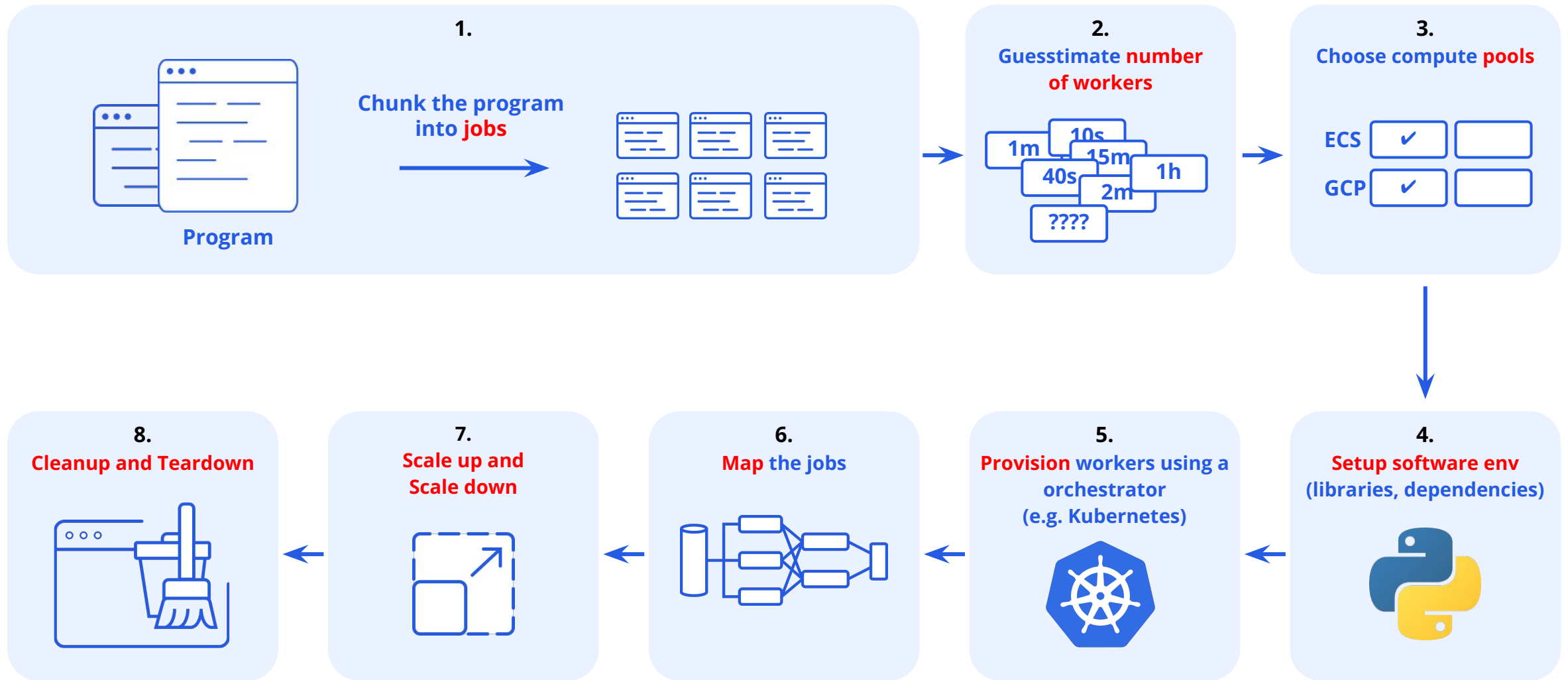
## **Map the tasks onto**

cores/machines/instances/grids/clouds

# Steps to Parallelize a Program

1. Chunk the program into **jobs**
2. Guesstimate the **workers** (cores) required to run the jobs - but APIs like AWS Batch might auto-provision.
3. Decide which compute **pools** to provision workers to run the jobs - Symphony, AWS EC2, OpenShift, etc.
4. Setup the **software** env (Libraries, Dependencies, etc.) for compute pools -
5. **Provision** workers on the pools using a **orchestrator** (e.g. EC2, Kubernetes, ..)
6. **Map** the jobs to the workers in **order**.
7. **Scaleup** and **Scaledown** as needed
8. Cleanup and **Teardown**

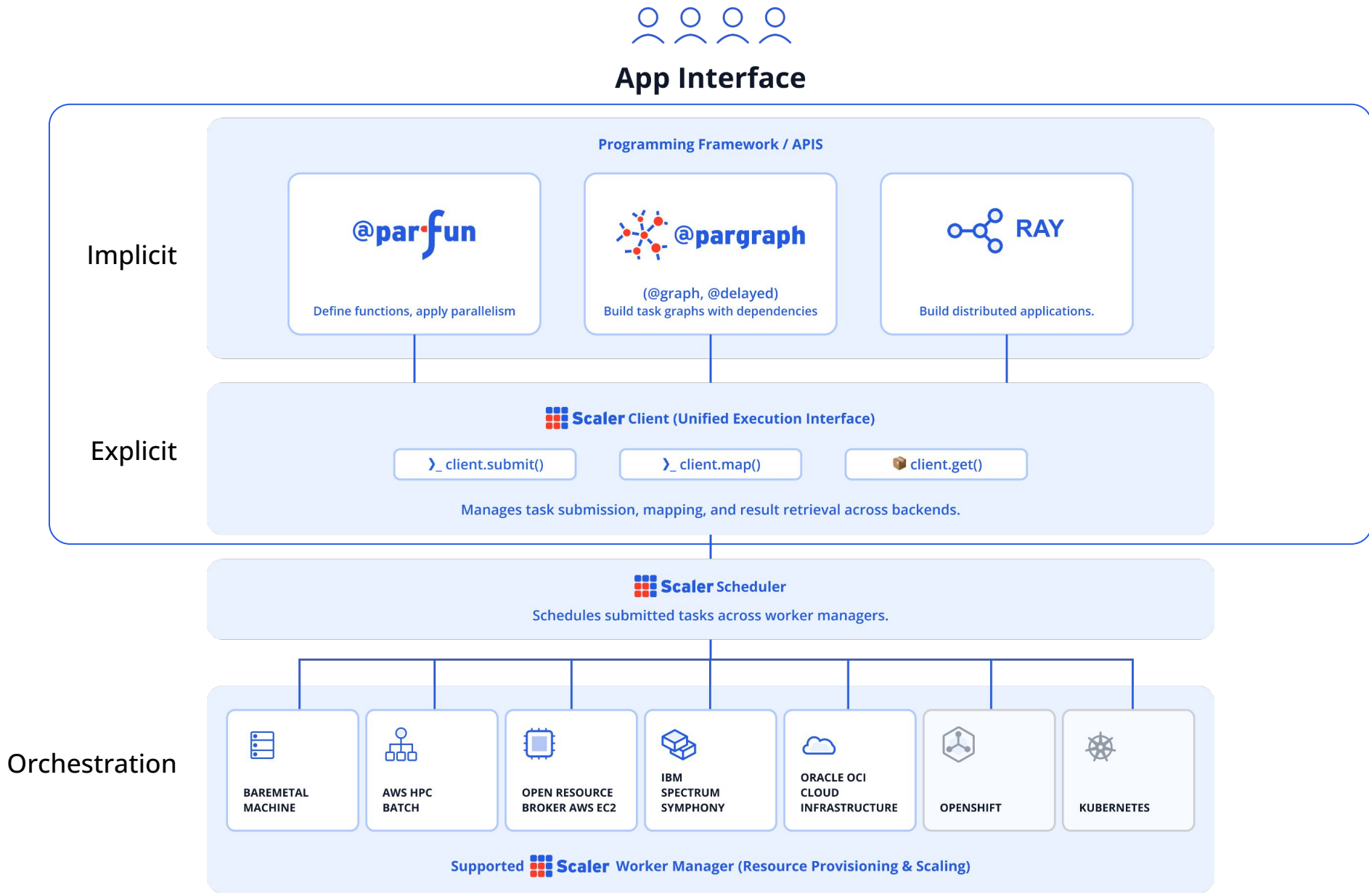
# HPC: Parallelizing a program



## Solution

Parallelization should be **implicit**, **automatic**, and **self-scaling**, with details like images and orchestration hidden from view.

# Scaler Solution



# Credit Valuation Adjustment (CVA) Example

Go to example [↗](#)

## Simple paragraph decorator

```
# — Survival probabilities & LGD —————
```

```
def compute_survival_probs(  
    credit_curves: List[CreditCurve],  
    scenarios: RateScenarios,  
    ) -> np.ndarray:  
    time_grid = scenarios.time_grid  
    n_cp = len(credit_curves)  
    n_t = len(time_grid)  
    surv = np.zeros((n_cp, n_t))  
    for i, cc in enumerate(credit_curves):  
        surv[i] = cc.survival_prob(time_grid)  
    return surv
```

```
def estimate_lgd(  
    cds_spreads: np.ndarray,  
    credit_curves: List[CreditCurve],  
    n_simulations: int = 5000,  
    seed: int = 99,  
    ) -> np.ndarray:  
    rng = np.random.default_rng(seed)  
    n_cp = len(credit_curves)  
    lgd_estimates = np.zeros(n_cp)
```

@delayed

Put an “@delayed” above  
functions for using paragraph



# Credit Valuation Adjustment (CVA) Example

[Go to example ↗](#)

Parfun decorator to make a function run in parallel


```
npvs = signs * (float_pv - fixed_pv)

# Positive exposure per counterparty
pos = np.maximum(npvs, 0.0) * alive
np.add.at(cp_exposure[step], cp_ids, pos)
```

```
return cp_exposure
```

```
def _price_scenario_batch(
    scenario_batch: List[np.ndarray],
    time_grid: np.ndarray,
    pay_times: np.ndarray,
    day_fracs: np.ndarray,
    fixed_rates: np.ndarray,
    notionals: np.ndarray,
    signs: np.ndarray,
    cp_ids: np.ndarray,
```

```
@pf.parallel(
    split=pf.per_argument(scenario_batch=pf.py_list.by_chunk),
    combine_with=pf.py_list.concat,
    fixed_partition_size=10,
)
```



# Performance statistics

Go to Example [↗](#)

Examples	Parfun	Pargraph	Client	Workers	Num Workers	Ratio: Speed/ Workers	Speedup	Sequential Runtime <div></div> Parallel Runtime <div></div>
Alpha Research	Yes	No	AWS	EC2	8	0.31	2.51	<div><div></div><div>14m 38.462s</div><div>5m 49.054s</div></div>
Implied Vol Surface	Yes	No	AWS	EC2	128	0.26	33	<div><div></div><div>81m 46.430s</div><div>2m 12.139s</div></div>
Swap CVA	Yes	Yes	AWS	EC2	64	0.43	27.6	<div><div></div><div>35m 12.430s</div><div>1m 16.413s</div></div>
Portfolio-Level CVA/XVA	No	Yes	NATIVE	NATIVE	16	0.64	10.27	<div><div></div><div>64m 3.8s</div><div>6m 14.1s</div></div>

# Scalerize this: Cloud Parallelization in 30 minutes

## One-time cloud env setup

Provide cloud credentials and perform

All in one config

Install and run command  
``scaler config.toml``

## Program Parallelization

Annotate application functions with parfun  
(data parallelization)

Annotate application functions with paragraph to parallelize the program  
(DAG Parallelization)

Connect Program to the scheduler and execute in context.

# Where to find Scaler and Documentation



GitHub



Documentation Site



Reach out for an invite to Slack

# Appendix

# Enterprise Compute: A spectrum of use cases

**Long running compute on thousands of cores  
(simulation, stress testing)**

## Case Study:

A market risk group at a large financial institution typically has grids that are used between the hours of 5P-11P and is mostly unused the rest of the day. These grids can cost more than \$50MM a year and the low utilization also has negative environmental impact.

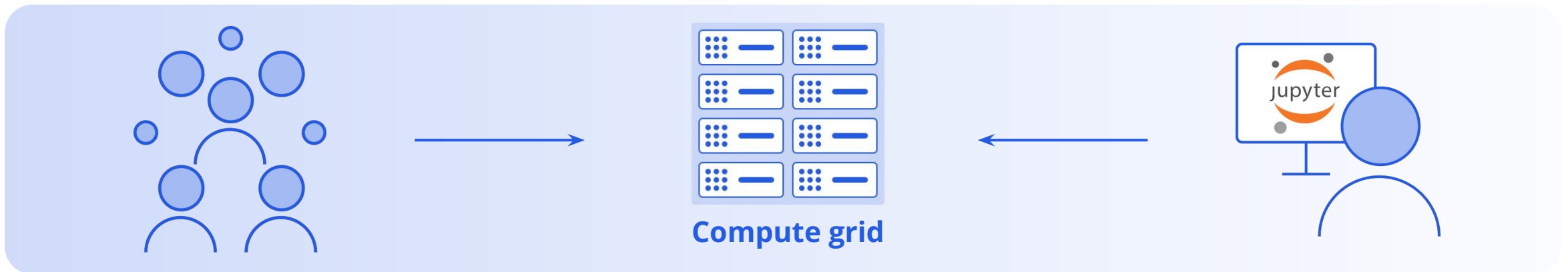
**Someone sitting at desk want access on the fly  
(model development?)**

## Case Study:

A quant in a modeling team wishes to run an ad-hoc pricing or analytical run on a portfolio. The quant has to spend several hours setting up the environment on AWS and provisioning the cores.

**Large compute**

**Small compute**



# What is being contributed

**openGRIS** — Open standard for Grid Resource Scheduling

 **Scaler** — Reference implementation

 **@pargraph** and **@parfun** — Client libraries for implicit parallelization

# Credit Valuation Adjustment (CVA) Example

Go to example [↗](#)

Scaler/pargraph/parfun imports and defining the remote addresses


## Native Python

```
from dataclasses import dataclass, field
from typing import Dict, List, Tuple

import numpy as np
import scipy.interpolate
import scipy.optimize

# — Domain types —
```

Add each time,  
only a few lines



## Paragraph + Parfun

```
from scaler import Client
from pargraph import GraphEngine, delayed, graph
import parfun as pf
scheduler_address = "tcp://172.31.3.107:7788"
object_storage_address = "tcp://172.31.3.107:7789"
```



# Credit Valuation Adjustment (CVA) Example

Go to example [↗](#)

## Running code with scaler

### Local machine

```
pipeline_kwargs = dict(
    swap_rates=swap_rates, tenors=tenors,
    swaption_vols=swaption_vols, swaption_expiries=swaption_expiries, swaption_tenors=swaption_tenors,
    cds_spreads=cds_spreads, cds_tenors=cds_tenors,
    swaps=swaps, n_counterparties=n_counterparties,
    n_scenarios=n_scenarios, n_steps=n_steps,
)

result = cva_pipeline(**pipeline_kwargs)
result.n_scenarios = n_scenarios
result.n_swaps = len(swaps)

print(f"Portfolio CVA: ${result.total_cva:,.2f}")
print(f"Number of counterparties: {result.n_counterparties}")
print(f"Number of swaps: {result.n_swaps}")
print(f"Number of scenarios: {result.n_scenarios}")
```

Submit your change  
to Scaler to run it in  
the cloud instead of  
locally

### Scaler

```
client = Client(scheduler_address, object_storage_address=object_storage_address)
engine = GraphEngine(client)

graph_obj = cva_pipeline.to_graph()
dict_graph, keys = graph_obj.to_dict(**pipeline_kwargs)
(result,) = engine.get(dict_graph, keys)
```

# Credit Valuation Adjustment (CVA) Example

[Go to example ↗](#)

Parallelize code using parfun using scaler as a backend

```
log_P0_grid = curve._interp(safe_grid)
zr0_grid = -log_P0_grid / safe_grid

a, sigma = hw.mean_reversion, hw.volatility

# Split scenarios into a list for parfun distribution
scenario_list = [rate_scenarios[s] for s in range(n_scenarios)]

results = _price_scenario_batch(
    scenario_list, time_grid,
    pay_times, day_fracs, fixed_rates, notionals, signs, cp_ids, n_pay,
    n_counterparties, a, sigma,
    log_P0, zr0, log_P0_grid, zr0_grid,
)

return np.array(results)
```

```
with pf.set_parallel_backend_context(
    "scaler_remote",
    scheduler_address=scheduler_address,
    object_storage_address=object_storage_address,
    n_workers=128,
):
```