About Me

Raphaël Vandon
he/him
Genesis: “Scheduler Performance Issue”

- 300 DAGs generated from one python file is OK
- 300 DAGs from 300 python files overwhelms the Scheduler
- WHY?
Experimenting to understand the problem

- Hypothesis: Scheduler is slow because it has many files to parse
- Experiment: Vary the number of dags

- Understand the results:
  - Parsing time: ~300ms/file (logging.dag_processor_log_target=stdout)
  - Done every 30s in 2 processes means >200 DAGs is too much
We already have solutions!

- Increase the parsing interval
  
  scheduler.min_file_process_interval=120

- More processes
  
  scheduler.parsing_processes=4

- Run the DAG processor separately
  
  scheduler.standalone_dag_processor=True
  airflow dag-processor
from airflow.decorators import dag, task
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.sensors.s3_key_sensor import S3KeySensor
from airflow.models import Variable
from airflow.utils.dates import days_ago

POSTGRES_CONN_ID = Variable.get("POSTGRES_CONN_ID", default_var="postgres_redshift")
S3_BUCKET = Variable.get("S3_BUCKET", default_var="my-dag-bucket")
S3_KEY = Variable.get("S3_KEY", default_var="dags/")
SCHEDULE_INTERVAL = Variable.get("SCHEDULE_INTERVAL", default_var="0 ** * * *")
DAG_RETRIES = int(Variable.get("DAG_RETRIES", default_var="3"))

def update_table_dag():
    s = S3KeySensor(task_id="check_s3", bucket_key=S3_KEY, wildcard_match=True, bucket_name=S3_BUCKET)
    t = PostgresOperator(task_id="query_t", sql="select * from my_table;", postgres_conn_id=POSTGRES_CONN_ID)
    s >> t

update_table_dag()
What is happening?

- 1 process per DAG parsed
- Working on a copy of the memory of the main process
- Imports are discarded when we’re done with the dag.

- Software optimisation is very often identifying what is repeated, and then finding a way to do it only once.
- We found what was repeated
- How to do it only once?
Solutions

- Communication between processes?
- Stop using 1 process per DAG?
- Do the imports in the main thread before forking?
Results

<table>
<thead>
<tr>
<th>Script</th>
<th>Avg ms on Main</th>
<th>Avg ms with Preload</th>
<th>Change %</th>
</tr>
</thead>
<tbody>
<tr>
<td>example_sftp_to_wasb</td>
<td>632</td>
<td>192</td>
<td>-70%</td>
</tr>
<tr>
<td>example_azure_blob_to_gcs</td>
<td>696</td>
<td>216</td>
<td>-69%</td>
</tr>
<tr>
<td>example_local_to_adls</td>
<td>536</td>
<td>186</td>
<td>-65%</td>
</tr>
<tr>
<td>example_adls_delete</td>
<td>510</td>
<td>190</td>
<td>-63%</td>
</tr>
<tr>
<td>example_local_to_wasb</td>
<td>509</td>
<td>197</td>
<td>-61%</td>
</tr>
<tr>
<td>example_adf_run_pipeline</td>
<td>577</td>
<td>226</td>
<td>-61%</td>
</tr>
<tr>
<td>example_azure_service_bus</td>
<td>555</td>
<td>283</td>
<td>-49%</td>
</tr>
<tr>
<td>example_postgres</td>
<td>360</td>
<td>184</td>
<td>-49%</td>
</tr>
<tr>
<td>example_azure_cosmosdb</td>
<td>355</td>
<td>197</td>
<td>-45%</td>
</tr>
<tr>
<td>example_docker</td>
<td>337</td>
<td>187</td>
<td>-44%</td>
</tr>
<tr>
<td>example_taskflow_api_docker_virtualenv</td>
<td>773</td>
<td>441</td>
<td>-43%</td>
</tr>
<tr>
<td>example_docker_copy_data</td>
<td>350</td>
<td>200</td>
<td>-43%</td>
</tr>
<tr>
<td>example_azure_container_instances</td>
<td>316</td>
<td>185</td>
<td>-41%</td>
</tr>
<tr>
<td>example_azure_synapse</td>
<td>321</td>
<td>189</td>
<td>-41%</td>
</tr>
<tr>
<td>example_docker_swarm</td>
<td>351</td>
<td>207</td>
<td>-41%</td>
</tr>
<tr>
<td>example_fileshare</td>
<td>327</td>
<td>193</td>
<td>-41%</td>
</tr>
<tr>
<td>example_zendesk_custom_get</td>
<td>315</td>
<td>193</td>
<td>-39%</td>
</tr>
<tr>
<td>example_ftp</td>
<td>311</td>
<td>198</td>
<td>-37%</td>
</tr>
<tr>
<td>example_sql_column_table_check</td>
<td>259</td>
<td>173</td>
<td>-33%</td>
</tr>
<tr>
<td>example_sql_execute_query</td>
<td>270</td>
<td>187</td>
<td>-31%</td>
</tr>
<tr>
<td>example_github</td>
<td>278</td>
<td>199</td>
<td>-29%</td>
</tr>
<tr>
<td>example_s3</td>
<td>1291</td>
<td>942</td>
<td>-27%</td>
</tr>
<tr>
<td>example_sagemaker</td>
<td>1339</td>
<td>997</td>
<td>-26%</td>
</tr>
<tr>
<td>example_http</td>
<td>263</td>
<td>196</td>
<td>-25%</td>
</tr>
</tbody>
</table>
About those usages of Variable...

- Not too slow when using a local DB
- ...but as soon as there are network calls, it’s bad

- Default settings: 300ms/DAG
- With AWS Secret Manager: 800ms/DAG
Solution: don’t do it?

- Don’t use Variables in top level DAG code
- It’s bad practice
- The documentation says you shouldn’t do it
- ...
- Users do it anyway
Adding a cache always solves everything

(no)

(but yes)
There is now an optional cache on Variables

- Uses Python’s `multiprocessing.Manager`
- `secrets.use_cache=true`
- Caveats!
- I can make arbitrarily good looking benchmarks

And Connections

Only during parsing

It’s experimental
Conclusion

- You don’t have to be an expert to have an impact
- Just time and motivation to investigate
Questions?

Get in contact:
vandonr@gmail.com
linkedin.com/in/vandonr
or on Airflow Slack