Optimising Airflow Performance

Tips & strategies to enhance metadata database performance

Pankaj Singh Software Engineer @ Astronomer & Airflow Committer

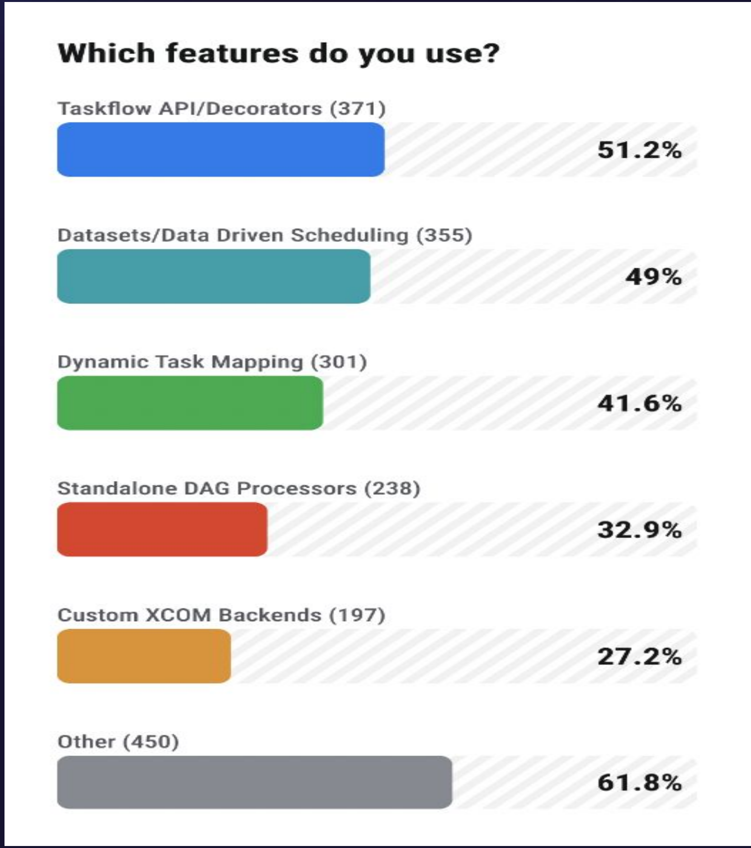
Pankaj Koti Software Engineer @ Astronomer & Airflow Committer

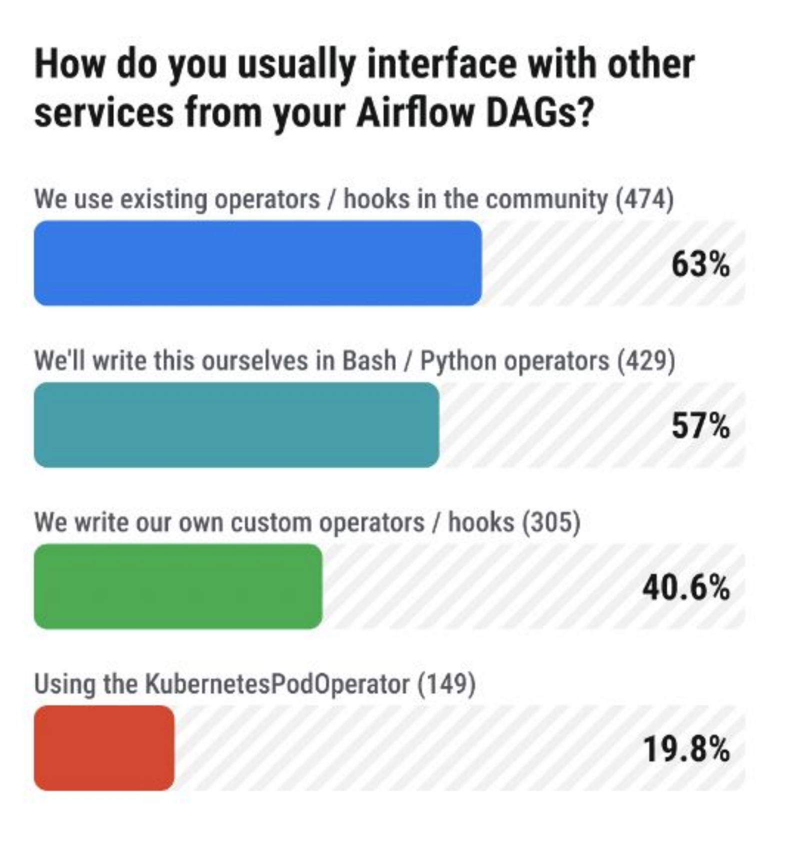
ASTRONOMER

What We'll Cover

1. DAG Authoring Best Practices
2. Database Optimization
    a. Unused Indexes
    b. Missing Indexes
    c. Table and Index Bloat

2023 Airflow Survey Result





DAG Authoring

Expensive Call

Avoid making network calls or performing heavy computations at the top level of the code.

Heavy Library Import

Avoid top-level imports for large libraries

Jinja Template

Use Jinja templates to access Airflow resources, as they are resolved at runtime

Detect Top-level Code

A simple test can be conducted by running python <dag_file>.py.

DAG Authoring

Example: Top-level expensive call

```python
import pendulum
from airflow import DAG
from airflow.decorators import task
from time import sleep

def expensive_api_call():
    print("Hello from Airflow!")
    sleep(5)  # Simulate an expensive call

# This call will be executed every time the DAG file is parsed
my_expensive_response = expensive_api_call()

with DAG(
    dag_id="example_bad_practice",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def print_expensive_api_call():
        print(my_expensive_response)

    print_expensive_api_call()
```

```
real    0m7.136s
user    0m2.030s
sys     0m0.104s
```

```python
import pendulum
from airflow import DAG
from airflow.decorators import task
from time import sleep

def expensive_api_call():
    sleep(5)  # Simulate an expensive call
    return "Hello from Airflow!"

with DAG(
    dag_id="example_good_practice",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def print_expensive_api_call():
        my_expensive_response = expensive_api_call()
        print(my_expensive_response)

    print_expensive_api_call()
```

```
real    0m2.283s
user    0m2.041s
sys     0m0.228s
```

DAG Authoring

Example: Top-level import

```python
import pendulum
from airflow import DAG
from airflow.decorators import task

# Expensive imports at the top level
import pandas as pd
import torch

with DAG(
    dag_id="example_bad_imports",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def process_data():
        # Use the imported libraries
        df = pd.DataFrame({'a': [1, 2, 3]})
        print(df)
        print(torch.__version__)

    process_data()
```

```
real    0m3.656s
user    0m4.213s
sys     0m0.213s
```

→

```python
import pendulum
from airflow import DAG
from airflow.decorators import task

with DAG(
    dag_id="example_good_imports",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def process_data():
        # Import the expensive libraries inside the task
        import pandas as pd
        import torch

        # Use the imported libraries
        df = pd.DataFrame({'a': [1, 2, 3]})
        print(df)
        print(torch.__version__)

    process_data()

```

```
real    0m2.267s
user    0m2.033s
sys     0m0.221s
```

DAG Authoring

Example: Jinja template

```python
from airflow import DAG
from airflow.decorators import task
from airflow.models import Variable
import pendulum

# Top-level variable fetch
foo = Variable.get('foo')

with DAG(
    dag_id="example_top_level_var_fetch",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def print_var():
        print(foo)

    print_var()
```

```
real    0m2.425s
user    0m2.120s
sys     0m0.295s
```

```python
from airflow import DAG
from airflow.decorators import task
import pendulum


with DAG(
    dag_id="example_jinja_template",
    schedule=None,
    start_date=pendulum.datetime(2024, 1, 1, tz="UTC"),
) as dag:

    @task
    def print_var(foo):
        print(foo)

    print_var("{{ var.value.get('foo') }}")

```

```
real    0m2.168s
user    0m2.027s
sys     0m0.140s
```

DAG Authoring

Example: Detect top-level code

`> python <my_dag.py>`	`> time python <my_dag.py>`

Optimising Database Performance

Metadata Performance Degradation

High Disk Consumption

Ex - The TI table uses about 4.7GB, but its indexes add another 20GB

Slow Query

Larger index sizes can significantly slow down queries by increasing disk I/O, lock contention, and resource usage

Scheduler Liveness Failure

Scheduler fails to respond, often due to metadatabase poor performance

Unused Indexes

Identifying Unused Indexes

Identifying unused indexes can be challenging due to various factors

Index Scan Metrics Over Time

How frequently the indexes are used in query execution over a period can change

Dynamic Usage Patterns

The indexes used vary depending on the specific use cases and feature requirements

Version-Related Usage

Indexes may be added or removed depending on the version of Airflow

Unused Indexes

```
SELECT
  schemaname || '.' || relname AS table,
  indexrelname AS index,
  pg_size_pretty(pg_relation_size(i.indexrelid)) AS index_size,
  idx_scan as index_scans
FROM
  pg_stat_user_indexes ui
JOIN
  pg_index i ON ui.indexrelid = i.indexrelid
WHERE
  NOT indisunique AND idx_scan = 0
```

Unused Indexes: Time Is Also a Factor

Index	12/04/2024	15/04/2024	17/04/2024
idx_last_scheduling_decision (size MB)	5047	5125	5183
idx_last_scheduling_decision (scan #)	0	0	0
idx_log_dag (size MB)	2291	2353	2399
idx_log_dag (scan #)	6	6	6

Deleting Unused Indexes

Deleted index

- Table: dag_run
- Index: idx_last_scheduling_decision

Airflow 2.9.2
https://github.com/apache/airflow/pull/39275

Potential candidate

- Table: log
- Index: idx_log_dag

Impact of Index Deletion

More disk space

Deleting the idx_last_scheduling_decision indexes freed up 5GB of disk space

Fast query

Improving the performance of write queries (such as insert and update)

Scheduler liveness failure

Gain acceptable performance to prevent frequent scheduler failures

Future: Exporting Stats for Index Usage

Reliable Index Stats

- To identify unused indexes in PostgreSQL, query the pg_stat_user_indexes view
- To export pg_stat_user_indexes via Prometheus, use the PostgreSQL exporter tool

```yaml
custom_queries:
  - query: |
      SELECT
        schemaname || '.' || relname AS table,
        indexrelname AS index,
        pg_size_pretty(pg_relation_size(i.indexrelid)) AS index_size,
        idx_scan as index_scans
      FROM
        pg_stat_user_indexes ui
      JOIN
        pg_index i ON ui.indexrelid = i.indexrelid
      WHERE
        NOT indisunique
    metrics:
      - table:
          usage: "LABEL"
          description: "Name of the table"
      - index:
          usage: "LABEL"
          description: "Name of the index"
      - index_size:
          usage: "LABEL"
          description: "Size of the index"
      - index_scans:
          usage: "COUNTER"
          description: "Number of index scans"
```

Missing Indexes

Slow Queries

Slow DAG List Page

Time take to loading dag list page was proportional to size of metadata.

Slow Stale Metadata Deletion

Stale metadata deletion of 1 week's data took 7 mins and Astronomer Support needed to delete data of 1 year for a customer which could take around 6hrs.

Slow Query Side Effect

Turnaround Time to Fetch Results

Time time taken to get the results increases.

High CPU Utilization

Sequential scanning of the metadata database raises CPU utilization

Identify Slow Query

```
# Add Extension
CREATE EXTENSION pg_stat_statements;


# Restart DB


# Get Stats
SELECT query,
       calls,
       min_time,
       max_time,
       mean_time,
       total_time
FROM   pg_stat_statements
ORDER  BY mean_time DESC;
```

Identify Slow Query - Result

_query	calls	min_time	mean_time	total_time
SELECT +	2	1.274807	1.654795	2.929602
substring(query, $1, $2) AS trimmed_qu				
SELECT +	1	1.157411	1.157411	1.157411
substring(query, $1, $2) AS _query,cal				
SELECT +	1	1.02273	1.02273	1.02273
substring(query, $1, $2) AS _query,cal				
SELECT dag_code.fileloc_hash, dag_code.fileloc, da	871	0.001832	0.721667	20.81991799999987
SELECT ab_permission.name, ab_view_menu.name AS na	43	0.14725	0.49942	7.8270149999999985
SELECT trigger.id +	2422	0.002959	0.47480500000000003	21.48129199999999
FROM trigger JOIN task_instance				
SELECT ab_user_1.id AS ab_user_1_id, ab_role.id AS	48	0.154585	0.452504	10.257408000000002
SELECT dag.dag_display_name, dag.dag_id, dag.root_	790	0.006876	0.301583	41.52062000000006
SELECT dag_priority_parsing_request.id, dag_priori	7658	0.0007080000000000001	0.284584	13.00931300000125
SELECT ab_role.id AS ab_role_id, ab_role.name AS a	165	0.002417	0.26421	13.192050000000004
SELECT a.attname, +	6	0.088876	0.225668	0.8616269999999999
pg_catalog.format_				
SELECT dag.dag_display_name, dag.dag_id, dag.root_	81	0.034542	0.195295	6.510171
SELECT c.relname FROM pg_class c JOIN pg_namespace	27	0.033126	0.17929299999999998	2.3051829999999995
SELECT ab_role.id, ab_role.name, ab_permission_vie	2	0.120209	0.16796	0.288169
SELECT anon_1.dag_display_name, anon_1.dag_id, ano	11	0.078249	0.163544	1.136466
SELECT dag.dag_display_name, dag.dag_id, dag.root_	2769	0.008291000000000001	0.155531	54.92970500000011
SELECT dag.dag_display_name, dag.dag_id, dag.root_	81	0.07234800000000001	0.143972	7.51676799999999
SELECT $1 AS anon_1 +	1357	0.001584	0.140863	14.92167799999982
FROM serialized_dag +				
WHERE se				
SELECT t.oid, typarray +	3722	0.004667	0.13902799999999998	40.376448000000025
FROM pg_type t JOIN pg_name				

Adding Missing Indexes

Table	Index
dag_tag	idx_dag_tag_dag_id
dag_warning	idx_dag_warning_dag_id
dag_schedule_dataset_reference	idx_dag_schedule_dataset_reference_dag_id
dag_schedule_dataset_reference	idx_dataset_dag_run_queue_target_dag_id
dag_schedule_dataset_reference	idx_task_outlet_dataset_reference_dag_id

Airflow 2.10
https://github.com/apache/airflow/pull/39638

Impact of Index Addition

Not all metadata index is require by everyone

Slow Stale metadata deletion

Improved by the addition of an index which reduced the time of deletion of 1 year data to 36sec from ~6 hours.

Future Works of a slow load of DAG list page

Split the page into multiple components that can execute parallel queries instead of the serial execution.

Table & Index Bloats

Causes, Detection, & Mitigation Strategies

ASTRONOMER

Introduction

What is a Database bloat?

Table Bloat

Excessive unused space in tables due to deleted or outdated data that hasn't been reclaimed

Common in systems with frequent updates & deletes

Index Bloat

Due to deleted or outdated index entries

Can significantly degrade performance as indexes grow larger than necessary

Causes of Table Bloats

Frequent Updates/Deletes

Lack of autovacuum or incorrect autovacuum settings

Lack of Proper Maintenance

Failure to regularly vacuum and analyze the database

Inefficient Storage

Over-allocated space during table creation or after significant changes in data volume

Credits:
https://hakibenita.com/postgresql-unused-index-size#index-and-table-bloat

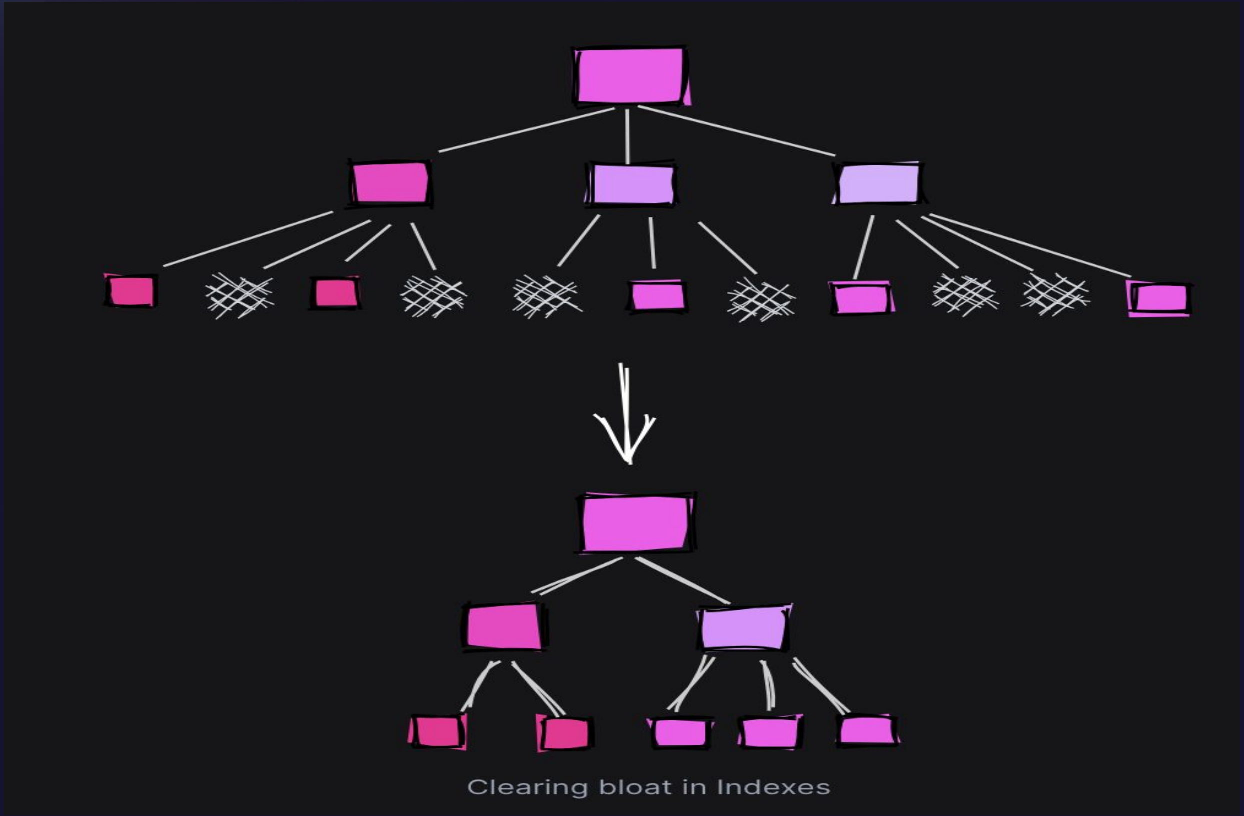
Causes of Index Bloats

Frequent Updates/Deletes on Indexed Columns

Indexes don't shrink automatically after deletions

Poor Index Management

Over-indexing and lack of regular index maintenance



Credits:
https://hakibenita.com/postgresql-unused-index-size#index-and-table-bloat

Impact of Table & Index Bloats

Performance Degradation

Slower query execution times

Increased I/O operations & memory usage

Increased Storage Costs

Larger than necessary database files

Maintenance Overhead

Longer backup & restore files

Detecting Bloats

Tools for Detection

PostgreSQL: pg_stat_all_tables, pgstattuple, pg_repack

MySQL: OPTIMIZE TABLE, ANALYZE TABLE

Key indicators

- Difference between table/index size and the actual data size
- Increasing table/index size without proportional data growth

Mitigating Table Bloats

Re-Create the Table

Often requires a lot of development, especially if the table is actively used as it's being rebuilt

Vacuum the Table

Query:

VACUUM FULL table_name

Will lock the page briefly

Using pg_repack

create EXTENSION pg_repack;

$ pg_repack -k --table table_name db_name

Mitigating Index Bloats

- Look for queries to detect index bloats based on your database
  E.g. For PostgreSQL, below is a helpful query
  https://github.com/ioguix/pgsql-bloat-estimation/blob/master/btree/btree_bloat.sql

- Reindex indexes with bloats

  REINDEX INDEX index_name

```sql
-- This query is compatible with PostgreSQL 8.2 and after
SELECT current_database(), nspname AS schemaname, tblname, idxname, bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_pct,
  fillfactor,
  CASE WHEN relpages > est_pages_ff
    THEN bs*(relpages-est_pages_ff)
    ELSE 0
  END AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_pct,
  is_na
  -- , 100-(pst).avg_leaf_density AS pst_avg_bloat, est_pages, index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples, relpages -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0 -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
      ) AS est_pages,
      coalesce(1 +
         ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/(100*(4+nulldatahdrwidth)::float))), 0
      ) AS est_pages_ff,
      bs, nspname, tblname, idxname, relpages, fillfactor, is_na
      -- , pgstatindex(idxoid) AS pst, index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples -- (DEBUG INFO)
  FROM (
      SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid, fillfactor,
            ( index_tuple_hdr_bm +
                maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
                  WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
                  ELSE index_tuple_hdr_bm%maxalign
                END
              + nulldatawidth + maxalign - CASE -- Add padding to the data to align on MAXALIGN
                  WHEN nulldatawidth = 0 THEN 0
                  WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
                  ELSE nulldatawidth::integer%maxalign
                END
            )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
            -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
      FROM (
          SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages,
              i.idxoid, i.fillfactor, current_setting('block_size')::numeric AS bs,
              CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
                WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64' THEN 8
                ELSE 4
              END AS maxalign,
              /* per page header, fixed size: 20 for 7.X, 24 for others */
              24 AS pagehdr,
              /* per page btree opaque data */
              16 AS pageopqdata,
              /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
              CASE WHEN max(coalesce(s.null_frac,0)) = 0
                  THEN 8 -- IndexTupleData size
                  ELSE 8 + (( 32 + 8 - 1 ) / 8) -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per index + 8 - 1 /8)
              END AS index_tuple_hdr_bm,
              /* data len: we remove null values save space using it fractionnal part from stats */
              sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS nulldatawidth,
              max( CASE WHEN i.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0 AS is_na
          FROM (
              SELECT ct.relname AS tblname, ct.relnamespace, ic.idxname, ic.attpos, ic.indkey, ic.indkey[ic.attpos], ic.reltuples, ic.relpages, ic.tbloid, ic.idxoid, ic.fillfactor,
                  coalesce(a1.attnum, a2.attnum) AS attnum, coalesce(a1.attname, a2.attname) AS attname, coalesce(a1.atttypid, a2.atttypid) AS atttypid,
                  CASE WHEN a1.attnum IS NULL
                  THEN ic.idxname
                  ELSE ct.relname
                  END AS attrelname
              FROM (
                  SELECT idxname, reltuples, relpages, tbloid, idxoid, fillfactor, indkey,
                      pg_catalog.generate_series(1,indnatts) AS attpos
                  FROM (
                      SELECT ci.relname AS idxname, ci.reltuples, ci.relpages, i.indrelid AS tbloid,
                          i.indexrelid AS idxoid,
                          coalesce(substring(
                              array_to_string(ci.reloptions, ' ')
                              from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor,
                          i.indnatts,
                          pg_catalog.string_to_array(pg_catalog.textin(
                              pg_catalog.int2vectorout(i.indkey)),' ')::int[] AS indkey
                      FROM pg_catalog.pg_index i
                      JOIN pg_catalog.pg_class ci ON ci.oid = i.indexrelid
                      WHERE ci.relam=(SELECT oid FROM pg_am WHERE amname = 'btree')
                      AND ci.relpages > 0
                  ) AS idx_data
              ) AS ic
              JOIN pg_catalog.pg_class ct ON ct.oid = ic.tbloid
              LEFT JOIN pg_catalog.pg_attribute a1 ON
                  ic.indkey[ic.attpos] <> 0
                  AND a1.attrelid = ic.tbloid
                  AND a1.attnum = ic.indkey[ic.attpos]
              LEFT JOIN pg_catalog.pg_attribute a2 ON
                  ic.indkey[ic.attpos] = 0
                  AND a2.attrelid = ic.idxoid
                  AND a2.attnum = ic.attpos
            ) i
            JOIN pg_catalog.pg_namespace n ON n.oid = i.relnamespace
            JOIN pg_catalog.pg_stats s ON s.schemaname = n.nspname
                                      AND s.tablename = i.attrelname
                                      AND s.attname = i.attname
            GROUP BY 1,2,3,4,5,6,7,8,9,10,11
      ) AS rows_data_stats
  ) AS rows_hdr_pdg_stats
) AS relation_stats
ORDER BY nspname, tblname, idxname;
```

Schema Name	Table Name	Index Name	Real Size (MB)	Extra Size (MB)	Extra (%)	Fill Factor	Bloat Size (MB)	Bloat (%)
airflow	dag_run	idx_last_scheduling_decision	8822	8789	99.62	90	8785.96	99.58
airflow	task_instance	task_instance_pkey	9928	7647	77.02	90	7374.58	74.27
airflow	xcom	idx_xcom_task_instance	9718	5190	53.41	90	4649.91	47.85
airflow	job	job_type_heart	5416	4725	87.24	90	4647.12	85.80
airflow	task_instance	ti_state_lkp	7275	4830	66.40	90	4559.35	62.67
airflow	task_instance	ti_job_id	2872	2292	79.81	90	2226.77	77.52
airflow	dag_run	idx_dag_run_dag_id	2245	2185	97.33	90	2178.83	97.03
airflow	job	idx_job_dag_id	2944	2253	76.53	90	2174.82	73.87
airflow	task_instance	ti_dag_run	3456	1856	53.72	90	1673.41	48.42
airflow	task_instance	ti_trigger_id	2159	1570	72.76	90	1505.84	69.75
airflow	xcom	xcom_pkey	4403	1550	35.20	90	1230.00	27.93
airflow	task_instance	ti_state_incl_start_date	2921	1157	39.62	90	954.65	32.67
airflow	task_instance	ti_dag_state	1686	761	45.14	90	655.26	38.86
airflow	task_instance	ti_pool	1637	712	43.50	90	606.38	37.04
airflow	log	idx_log_dag	4285	930	21.71	90	546.12	12.74
airflow	job	job_pkey	871	488	55.98	90	445.04	51.05
airflow	task_instance	idx_laminar_ti_end_date	976	388	39.77	90	323.21	33.10
airflow	xcom	idx_xcom_key	1554	387	24.95	90	258.84	16.65
airflow	task_instance	ti_state	674	254	37.69	90	207.27	30.71
airflow	job	idx_job_state_heartbeat	798	262	32.81	90	202.72	25.38
airflow	dag_run	dag_run_dag_id_execution_date_key	259	185	71.76	90	177.61	68.53
airflow	dag_run	dag_run_dag_id_run_id_key	309	182	59.08	90	168.19	54.39
airflow	log	idx_log_event	2531	398	15.75	90	162.70	6.43
airflow	dag_run	dag_run_pkey	120	87	72.48	90	83.91	69.40
airflow	dag_run	idx_laminar_dagrun_end_date	104	57	55.41	90	52.66	50.47
airflow	dag_run	dag_id_state	117	44	37.92	90	36.33	30.82
airflow	log	idx_log_dttm	1700	175	10.31	90	4.43	0.26
airflow	log	log_pkey	1700	175	10.31	90	4.42	0.26

Caveat: Do It at Your Own Risk

- Airflow metadata database expects the DDLs to be unaltered and only be modified via the migrations
- But if you're cautious and can take care of any potential conflicts then you're good to apply your findings and solutions

Thank you!

Questions?

#airflow-performance