Announcing pg_statviz
Announcing pg_statviz

- A minimalist extension and utility pair
- Time series analysis and visualization of PostgreSQL internal statistics
PostgreSQL internal statistics

- The **Cumulative Statistics System** (FKA Statistics Collector)
  - Postgres subsystem that collects info about system activity
- Dynamic statistics (right now)
- Cumulative statistics, but can be reset
- Table/index information on row & disk block levels
- This info can be reported via views
Motivation

- Why?
  - Track PostgreSQL performance over time and potentially perform tuning or troubleshooting
- Yes, but why?
  - So that people can understand their system better at a glance 👀
Motivation

• Working with customers
  – Who often have no idea how their database is performing
  – Or why it’s not working well
• Their monitoring tools don’t give them insights
How?

• Created for:
  – Snapshotting **cumulative and dynamic statistics**
  – Performing **time series analysis** on them

• Utility can produce visualizations for selected time ranges on the stored stats snapshots
Design Philosophy

- **K.I.S.S.** and **UNIX** philosophies
- Tool aims to be:
  - Modular
  - Minimal
  - Unobtrusive
- Does only what it's meant for: create snapshots of PostgreSQL statistics for visualization and analysis.
Design Philosophy

- Not for live monitoring displays
  - But one could...
- Open schema, clearly defined
  - Data easily exportable
- No built-in scheduler
- No built-in data retention policy mechanism
Design

- Components
  - PostgreSQL extension
  - Python utility for retrieving stored snapshots & creating simple visualizations using Matplotlib
- Nothing to put in `shared_preload_libraries`
- No need to restart Postgres
Installation

- Extension installation:
  
  ![CREATE EXTENSION pg_statviz;](image)

- Utility installation:

  $ pip install pg_statviz
Usage

- Extension can be used by superusers, or any user that has **pg_monitor** role privileges
- To take a snapshot, e.g. from **psql**:
  ```sql
  SELECT pgstatviz.snapshot();
  ```
Usage

Announcing pg_statviz

$ pg_statviz --help
[-D FROM TO] [-O OUTPUTDIR]
  {analyze, buf, cache, checkpoint, conn, tuple, wait, wal} ...

run all analysis modules

positional arguments:
  {analyze, buf, cache, checkpoint, conn, tuple, wait, wal}
  analyze  run all analysis modules
  buf      run buffers written analysis module
  cache    run cache hit ratio analysis module
  checkpoint  run checkpoint analysis module
  conn     run connection count analysis module
  tuple    run tuple count analysis module
  wait     run wait events analysis module
  wal      run WAL generation analysis module

options:
  --help
  --version  show program's version number and exit
  -d DBNAME, --dbname DBNAME  database name to analyze (default: 'vyruss')
  -h HOSTNAME, --host HOSTNAME  database server host or socket directory (default: '/var/run/postgresql')
  -p PORT, --port PORT  database server port (default: '5432')
  -U USERNAME, --username USERNAME  database user name (default: 'vyruss')
  -W, --password  force password prompt (should happen automatically) (default: False)
  -D FROM TO, --daterange FROM TO  date range to be analyzed in ISO 8601 format e.g. 2023-01-01T00:00
                                 2023-01-01T23:59 (default: [])
  -O OUTPUTDIR, --outdir OUTPUTDIR  output directory (default: .)
Usage

```
vyruss@rancor: $ pg_statviz -d faf -U postgres -D 2023-01-17T23:00 2024-01-01
INFO:pg_statviz.modules.buf:Running buffers written analysis
INFO:pg_statviz.modules.buf:Saving pg_statviz_rancor_5432_buf.png
INFO:pg_statviz.modules.buf:Saving pg_statviz_rancor_5432_buf_rate.png
INFO:pg_statviz.modules.checkp:Running checkpoint analysis
INFO:pg_statviz.modules.checkp:Saving pg_statviz_rancor_5432_checkp.png
INFO:pg_statviz.modules.checkp:Saving pg_statviz_rancor_5432_checkp_rate.png
INFO:pg_statviz.modules.cache:Running cache hit ratio analysis
INFO:pg_statviz.modules.cache:Saving pg_statviz_rancor_5432_cache.png
INFO:pg_statviz.modules.conn:Running connection count analysis
INFO:pg_statviz.modules.conn:Saving pg_statviz_rancor_5432_conn_status.png
INFO:pg_statviz.modules.conn:Saving pg_statviz_rancor_5432_conn_user.png
INFO:pg_statviz.modules.tuple:Running tuple count analysis
INFO:pg_statviz.modules.tuple:Saving pg_statviz_rancor_5432_tuple.png
INFO:pg_statviz.modules.wait:Running wait events analysis
INFO:pg_statviz.modules.wait:Saving pg_statviz_rancor_5432_wait.png
INFO:pg_statviz.modules.wal:Running WAL generation analysis
INFO:pg_statviz.modules.wal:Saving pg_statviz_rancor_5432_wal.png
INFO:pg_statviz.modules.wal:Saving pg_statviz_rancor_5432_wal_rate.png
vyruss@rancor: $  
```
Modules

- **analyze** (default) - run all analysis modules
- **buf** - buffers written, buffer write rate
- **cache** - cache hit ratio
- **checkp** - checkpoint analysis, checkpoint rate
- **conn** - connection count, by status and by user
- **tuple** - tuple count analysis
- **wait** - wait events analysis
- **wal** - WAL generation analysis
Obligatory

- And yes, it has a logo :)
Announcing pg_statviz
Announcing pg_statviz
Announcing pg_statviz
Use cases

- "Black box" database
  - Deploy and let the developers wreak havoc
  - Identify users/components
- Performance troubleshooting
- Observe and monitor DB behaviour over a long period
  - During a stress test run
  - 8 hours (working hours) / 24 hours (complete day cycle)
  - A month / years (?)
Extension implementation

- The basis of everything

```sql
CREATE TABLE IF NOT EXISTS @extschema@.snapshots(
    snapshot_tstamp timestamp tzn PRIMARY KEY
);
```
Extension implementation

- Everything follows from that, code is modular

```sql
CREATE TABLE IF NOT EXISTS @extschema@.buf(
    snapshot_tstamp timestamp(tz) REFERENCES @extschema@.snapshots(snapshot_tstamp) ON DELETE CASCADE PRIMARY KEY,
    checkpoints_timed bigint,
    checkpoints_req bigint,
    checkpoint_write_time double precision,
    checkpoint_sync_time double precision,
    buffers_checkpoint bigint,
    buffers_clean bigint,
    maxwritten_clean bigint,
    buffers_backend bigint,
    buffers_backend_fsync bigint,
    buffers_alloc bigint,
    stats_reset timestamp(tz))
```
### Extension implementation

```sql
faf=> \dt pgstatviz.*

List of relations

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Type</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgstatviz</td>
<td>buf</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>conf</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>conn</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>db</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshots</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>wait</td>
<td>table</td>
<td>postgres</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>wal</td>
<td>table</td>
<td>postgres</td>
</tr>
</tbody>
</table>

(7 rows)
```
Announcing pg_statviz

• Snapshot function

```sql
-- Snapshots
CREATE OR REPLACE FUNCTION @extschema@.snapshot()
RETURNS timestamp
AS $$
    DECLARE ts timestamp;
    BEGIN
        ts := clock_timestamp();
        INSERT INTO @extschema@.snapshots VALUES (ts);
        PERFORM @extschema@.snapshot_buf(ts);
        PERFORM @extschema@.snapshot_conf(ts);
        PERFORM @extschema@.snapshot_conn(ts);
        PERFORM @extschema@.snapshot_db(ts);
        PERFORM @extschema@.snapshot_wait(ts);
        PERFORM @extschema@.snapshot_wal(ts);
        RAISE NOTICE 'created pg_statviz snapshot';
    RETURN ts;
$$ LANGUAGE PLPGSQL;
```
Announcing pg_statviz

Extension implementation

- Buffers

```sql
CREATE OR REPLACE FUNCTION @extschema@.snapshot_buf(snapshot_tstamp timestamptz) RETURNS void AS $$
    INSERT INTO @extschema@.buf (snapshot_tstamp, checkpoints_timed, checkpoints_req, checkpoint_write_time, checkpoint_sync_time, buffers_checkpoint, buffers_clean, maxwritten_clean, buffers_backend, buffers_backend_fsync, buffers_alloc, stats_reset)
    SELECT snapshot_tstamp, checkpoints_timed, checkpoints_req, checkpoint_write_time, checkpoint_sync_time, buffers_checkpoint, buffers_clean, maxwritten_clean, buffers_backend, buffers_backend_fsync, buffers_alloc, stats_reset
    FROM pg_stat_bgel;
$$ LANGUAGE SQL;
```
Extension implementation

- Connections

```
CREATE OR REPLACE FUNCTION @extschema@.snapshot_conn(snapshot_tstamp timestamp timestamptz) RETURNS void AS $$
WITH
    pgsa AS (SELECT * FROM pg_stat_activity
              FROM current_database()
              WHERE state IS NOT NULL),
    userconns AS (SELECT jsonb_agg(uc) FROM (SELECT username AS user, count(*) AS connections
                                              FROM pgsa
                                              GROUP BY username) uc)
    INSERT INTO @extschema@.conn (snapshot_tstamp, conn_total, conn_active, conn_idle, conn_idle_trans, conn_idle_trans_abort, conn_fastpath, conn_users)
    SELECT snapshot_tstamp, count(*) AS conn_total,
           count(*) FILTER (WHERE state = 'active') AS conn_active,
           count(*) FILTER (WHERE state = 'idle') AS conn_idle,
           count(*) FILTER (WHERE state = 'idle in transaction') AS conn_idle_trans,
           count(*) FILTER (WHERE state = 'idle in transaction (aborted)') AS conn_idle_trans_abort,
           count(*) FILTER (WHERE state = 'fastpath function call') AS conn_fastpath,
           (SELECT * FROM userconns) AS conn_users
    FROM pgsa;
$$ LANGUAGE SQL;
```
Announcing pg_statviz

```
faf=> \df pgstatviz.*
```

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Result data type</th>
<th>Argument data types</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgstatviz</td>
<td>delete_snapshots</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot</td>
<td>timestamp with time zone</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_buf</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_conf</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_conn</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_db</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_wait</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
<tr>
<td>pgstatviz</td>
<td>snapshot_wal</td>
<td>void</td>
<td>snapshot_tstamp timestamp with time zone</td>
<td>func</td>
</tr>
</tbody>
</table>

(8 rows)
Utility implementation

- Modular code in Python

```python
import sys
from argh import ArghParser
from pg_statviz.modules.analyze import analyze
from pg_statviz.modules.buf import buf
from pg_statviz.modules.cache import cache
from pg_statviz.modules.checkp import checkp
from pg_statviz.modules.conn import conn
from pg_statviz.modules.tuple import tuple
from pg_statviz.modules.wait import wait
from pg_statviz.modules.wal import wal

# Python version check
if sys.version_info < (3, 7):
    raise SystemExit("This program requires Python 3.7 or later")
```
Utility implementation

- "Buffers written" data retrieval and preparation

```python
# Retrieve the snapshots from DB
cur = conn.cursor()
cur.execute(""""SELECT buffers_checkpoint, buffers_clean, buffers_backend, 
    stats_reset, snapshot_tstamp 
    FROM pgstatviz.buf 
    WHERE snapshot_tstamp BETWEEN %s AND %s 
    ORDER BY snapshot_tstamp""",
    (daterange[0], daterange[1])))
data = cur.fetchall(MAX_RESULTS)

if not data:
    raise SystemExit("No pg_statviz snapshots found in this database")

timestamps = [t['snapshot_tstamp'] for t in data]
blocksize = int(info['block_size'])

# Gather buffers and convert to GB
total = round((b['buffers_checkpoint']
               + b['buffers_clean']
               + b['buffers_backend'])
             * blocksize / 1073741824, 1) for b in data]
checkpoints = [round(b['buffers_checkpoint']
                     * blocksize / 1073741824, 1) for b in data]
bgwriter = [round(b['buffers_clean']
                  * blocksize / 1073741824, 1) for b in data]
backends = [round(b['buffers_backend']
                 * blocksize / 1073741824, 1) for b in data]
```
Utility implementation

- "Buffers written rate" data preparation

```python
# Buffer diff generator - yields 3-tuple list of the 3 rates in buffers/s
def bufdiff(data):
    yield (numpy.nan, numpy.nan, numpy.nan)
    for i, item in enumerate(data):
        if i + 1 < len(data):
            if data[i + 1]['stats_reset'] == data[i]['stats_reset']:
                s = (data[i + 1]['snapshot_timestamp'] - data[i]['snapshot_timestamp']).total_seconds()
                yield ((data[i + 1]['buffers_checkpoint'] - data[i]['buffers_checkpoint']) / s,
                       (data[i + 1]['buffers_clean'] - data[i]['buffers_clean']) / s,
                       (data[i + 1]['buffers_backend'] - data[i]['buffers_backend']) / s)
        else:
            yield (numpy.nan, numpy.nan, numpy.nan)

buffers = list(bufdiff(data))

# Normalize and round the rate data
total = [round((b[0] + b[1] + b[2]) * blkysz / 1048576, 1 if b[0] >= 100 else 2) for b in buffers]
checkpoints = [round(b[0] * blkysz / 1048576, 1 if b[0] >= 100 else 2) for b in buffers]
bgwriter = [round(b[1] * blkysz / 1048576, 1 if b[0] >= 100 else 2) for b in buffers]
backends = [round(b[2] * blkysz / 1048576, 1 if b[0] >= 100 else 2) for b in buffers]
```
Utility implementation

- Plotting

```python
# Plot buffer rates
plt, fig = plt.plot_setup()
plt.suptitle(f"pg_statviz :: {info['hostname']}:{port}"),
fontweight='semibold')
plt.title("Buffer write rate")
plt.plot_date(tstamps, total, label="total", aa=True,
linestyle='solid')
plt.plot_date(tstamps, checkpoints, label="checkpoints", aa=True,
linestyle='solid')
plt.plot_date(tstamps, bgwriter, label="bgwriter", aa=True,
linestyle='solid')
plt.plot_date(tstamps, backends, label="backends", aa=True,
linestyle='solid')
plt.xlabel("Timestamp", fontweight='semibold')
plt.ylabel("Avg. write rate in MB/s", fontweight='semibold')
fig.legend()
fig.tight_layout()
outfile = f"{outputdir.rstrip("/") + "/" if outputdir
else ''}pg_statviz_{info['hostname']}
.replace("/", "-\})_{port}_buf_rate.png"
_logger.info(f"Saving {outfile}")
plt.savefig(outfile)
```
Utility implementation

- Some charts are not so easy (wait events)
• Easy to change matplotlib settings

```python
def setup():
    for f in ["NotoSans-Regular.ttf", "NotoSans-SemiBold.ttf"]:  
        f = importlib.resources.path("pg_statviz.libs", f)  
        fnt.fontManager.addfont(f)  
        plt.rcParams["font.family"] = 'Noto Sans'  
        plt.rcParams["font.size"] = 12  
    im = plt.imread(importlib.resources.path("pg_statviz.libs",  
                                            "pg_statviz.png"))  
    height = im.shape[0]  
    fig = plt.figure(figsize=(19.2, 10.8))  
    fig.figimage(im, 5, (fig.bbox.ymax - height - 6), zorder=3)  
    plt.grid(visible=True)  
    plt.ticklabel_format(axis='y', style='plain')  
    plt.gcf().autofmt_xdate()  
    return plt, fig
```
The Future

- Code is currently at "feature-complete alpha" maturity
- Needs:
  - Packaging for PGDG repositories and Linux distributions
  - Additional modules for stats to monitor (such as replication, I/O (pg_stat_io), tables/indexes)
  - Data management/retention functions
  - Writing of Python regression tests
The Future

- Google Summer of Code PostgreSQL Project
Thank you!

- Project: https://github.com/vyruss/pg_statviz
- You can find me on Mastodon: @vyruss@fosstodon.org
Thank you!