What’s Missing For Postgres Monitoring

@LukasFittl
What are the problems with Postgres monitoring?
It’s incomplete.
It’s hard to access & understand.
It contains sensitive information.
It’s incomplete.

It’s hard to access & understand.

It contains sensitive information.
Connection Handling

pg_stat_activity

Log events

Connection received
Disconnection
Incomplete startup packet (client failed to connect)
Could not receive data from client / connection to client lost
EOF on client connection with an open transaction
Terminating connection due to administrator command
Remaining connection slots are reserved for superuser (out of connections)
Too many connections for role
Could not accept SSL connection
Unsupported frontend protocol
Incomplete message from client
Too many connections for database
Client-side connection latency

What’s Missing

Hard to track from the Postgres server side
- could libpq have built-in measurements here?
- should \texttt{timing} in psql give connection time and planning/execution time separately?
Connection Security

pg_stat_ssl
pg_stat_gssapi

Log events
Connection authorized
Authentication failed / pg_hba.conf rejects connection
Aggregation of security-relevant Postgres events

Which IPs logged in as superuser?

How many login failures occurred recently?

Which of my pg_hba lines are matching?
Query Planning

EXPLAIN

New EXPLAIN: Buffers for Planning

New pg_stat_statements planning time

Log events

auto_explain
EXPLAIN: Buffers for Planning

QUERY PLAN

Limit (cost=0.00..0.03 rows=1 width=86) (actual time=0.446..0.446 rows=1 loops=1)
 Buffers: shared read=1
  -> Append (cost=0.00..103559.88 rows=3406392 width=86) (actual time=0.445..0.446 rows=1 loops=1)
    Buffers: shared read=1
      -> Seq Scan on query_stats_hourlies_60d_20200127 query_stats_hourlies_60d_1 (cost=0.00..527.90 rows=20790 width=86) (actual time=0.440..0.440 rows=1 loops=1)
        Buffers: shared read=1
      -> Seq Scan on query_stats_hourlies_60d_20200128 query_stats_hourlies_60d_2 (cost=0.00..723.93 rows=28493 width=86) (never executed)

Planning Time: 45.882 ms
 Buffers: shared hit=8306 read=435 dirtied=10
Execution Time: 0.446 ms
(128 rows)
### New in Postgres 13

**pg_stat_statements: Planning Time**

```sql
=# SELECT queryid, substring(query for 40), mean_exec_time, mean_plan_time, max_plan_time FROM pg_stat_statements ORDER BY mean_plan_time DESC LIMIT 5;
```

<table>
<thead>
<tr>
<th>queryid</th>
<th>substring</th>
<th>mean_exec_time</th>
<th>mean_plan_time</th>
<th>max_plan_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>586048399314747810</td>
<td>WITH upsert(backend_id, server_id, ident</td>
<td>0.440361</td>
<td>5.890649</td>
<td>5.890649</td>
</tr>
<tr>
<td>5426874022189006220</td>
<td>WITH data(table_id, name, first_snapshot</td>
<td>18.846979</td>
<td>5.452164</td>
<td>5.452164</td>
</tr>
<tr>
<td>3576712877697568576</td>
<td>WITH data(table_id, name, first_snapshot</td>
<td>17.85431479746835</td>
<td>5.032493797468352</td>
<td>12.714722</td>
</tr>
<tr>
<td>-1758450264182311255</td>
<td>WITH data(table_id, name, first_snapshot</td>
<td>17.870344956521738</td>
<td>4.544071499999999</td>
<td>6.236185</td>
</tr>
<tr>
<td>-1076182304104233502</td>
<td>WITH data(table_id, name, first_snapshot</td>
<td>15.446047395348836</td>
<td>3.378207406976743</td>
<td>5.378551</td>
</tr>
</tbody>
</table>
```

(5 rows)
Aggregate Plan Statistics

Many experimental Postgres extensions (pg_stat_plans, pg_store_plans, pg_stat_sql_plans, etc)

Not production ready, or merge-able into Postgres core
Active Queries

`pg_stat_activity`

`(state, query_start, xact_start, wait events)`
Additional & renamed wait events

Report wait event for cost-based vacuum delay.
Add description about LogicalRewriteTruncate wait event into document.
Add description about GSSOpenServer wait event into document.
Correct the descriptions of recovery-related wait events in docs.
Rename the recovery-related wait events.
Add wait events for WAL archive and recovery pause.
Add wait events for recovery conflicts.
Report missing wait event for timeline history file.
Report time spent in posix_fallocate() as a wait event.
Drop the redundant "Lock" suffix from LWLock wait event names.
Mop-up for wait event naming issues.
Breakdown of non-waiting active state
postgres=# SELECT state, wait_event_type, wait_event, substring(query for 100) FROM pg_stat_activity WHERE backend_type = 'client backend';

<table>
<thead>
<tr>
<th>state</th>
<th>wait_event_type</th>
<th>wait_event</th>
<th>substring</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td></td>
<td>COPY public.log_lines_30d_20200516 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l</td>
<td></td>
</tr>
<tr>
<td>active</td>
<td></td>
<td>COPY public.log_lines_30d_20200514 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l</td>
<td></td>
</tr>
<tr>
<td>active</td>
<td></td>
<td>COPY public.log_lines_30d_20200517 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l</td>
<td></td>
</tr>
<tr>
<td>active</td>
<td></td>
<td>COPY public.log_lines_30d_20200515 (log_line_id, server_id, backend_pid, occurred_at, log_file_id, l</td>
<td></td>
</tr>
<tr>
<td>active</td>
<td></td>
<td>SELECT state, wait_event_type, wait_event, substring(query for 100) FROM pg_stat_activity WHERE back</td>
<td></td>
</tr>
<tr>
<td>idle</td>
<td>Client</td>
<td>ClientRead</td>
<td></td>
</tr>
</tbody>
</table>

(6 rows)
perf top -g

Samples: 379K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 55672843733 lost: 0/0 drop: 15165/199698

<table>
<thead>
<tr>
<th>Children</th>
<th>Self</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 58.01%</td>
<td>0.91%</td>
<td>postgres</td>
<td>[.] CopyFrom</td>
</tr>
<tr>
<td>+ 46.72%</td>
<td>1.54%</td>
<td>postgres</td>
<td>[.] NextCopyFrom</td>
</tr>
<tr>
<td>+ 23.68%</td>
<td>0.98%</td>
<td>postgres</td>
<td>[.] InputFunctionCall</td>
</tr>
<tr>
<td>+ 20.72%</td>
<td>5.71%</td>
<td>postgres</td>
<td>[.] NextCopyFromRawFields</td>
</tr>
<tr>
<td>+ 15.13%</td>
<td>0.03%</td>
<td>perf</td>
<td>[.] __ordered_events__flush.part.0</td>
</tr>
<tr>
<td>+ 15.08%</td>
<td>0.03%</td>
<td>perf</td>
<td>[.] deliver_event</td>
</tr>
<tr>
<td>+ 14.73%</td>
<td>0.02%</td>
<td>perf</td>
<td>[.] hist_entry_iter__add</td>
</tr>
<tr>
<td>+ 11.81%</td>
<td>0.82%</td>
<td>perf</td>
<td>[.] iter_add_next_cumulative_entry</td>
</tr>
<tr>
<td>+ 11.60%</td>
<td>0.45%</td>
<td>postgres</td>
<td>[.] timestamp_in</td>
</tr>
<tr>
<td>+ 8.77%</td>
<td>0.99%</td>
<td>postgres</td>
<td>[.] DecodeDateTime</td>
</tr>
<tr>
<td>+ 8.36%</td>
<td>0.22%</td>
<td>[kernel]</td>
<td>[k] do_syscall_64</td>
</tr>
<tr>
<td>+ 7.29%</td>
<td>0.27%</td>
<td>[kernel]</td>
<td>[k] __softirqentry_text_start</td>
</tr>
<tr>
<td>+ 6.97%</td>
<td>0.01%</td>
<td>[kernel]</td>
<td>[k] net_rx_action</td>
</tr>
<tr>
<td>+ 6.54%</td>
<td>0.01%</td>
<td>[kernel]</td>
<td>[k] ena_io_poll</td>
</tr>
<tr>
<td>+ 6.13%</td>
<td>0.00%</td>
<td>libc-2.31.so</td>
<td>[.] __libc_start_main</td>
</tr>
</tbody>
</table>
### perf top -g

Samples: 379K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 55672843733 lost: 0/0 drop: 15165/199698

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Children</th>
<th>Self</th>
<th>Shared Object</th>
<th>Time Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[.] CopyFrom</td>
<td>58.01%</td>
<td>0.91%</td>
<td>postgres</td>
<td>NextCopyFrom</td>
</tr>
<tr>
<td>[.] CopyFrom</td>
<td>7.70%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>11.96%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFromRawFields</td>
<td>15.40%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[.] InputFunctionCall</td>
<td>11.65%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>11.65%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>15.13%</td>
<td>0.03%</td>
<td>perf</td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>15.08%</td>
<td>0.03%</td>
<td>perf</td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>14.73%</td>
<td>0.02%</td>
<td>perf</td>
<td></td>
</tr>
<tr>
<td>[.] NextCopyFrom</td>
<td>11.81%</td>
<td>0.83%</td>
<td>perf</td>
<td></td>
</tr>
</tbody>
</table>
Query Progress Monitoring
Historic Queries

pg_stat_statements

Log Events

Slow query (log_min_duration_statement)
Statement notice (log_statement)
auto_explain
Better handling of IN(...) lists & other ORM patterns
What's Missing

Linking pg_stat_statements with other views & logs
Finding queries based on application requests/customers

pg_stat_statements has no way of differentiating queries beyond the queried
Which customers were affected by a slow query?

What was the EXPLAIN plan for a SQL query involved in a particular slow web request?
Solution for per-customer analysis:

“citus_stat_statements” in Citus extension

```
SELECT partition_key as tenant_id,
       count(*) as tenant_unique_queries,
       sum(calls) as tenant_total_queries,
       sum(total_time) as total_query_time
FROM citus_stat_statements
WHERE partition_key is not null
GROUP BY tenant_id
ORDER BY tenant_total_queries DESC
LIMIT 5;
```

<table>
<thead>
<tr>
<th>tenant_id</th>
<th>tenant_unique_queries</th>
<th>tenant_total_queries</th>
<th>total_query_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>148</td>
<td>159295</td>
<td>753142.54</td>
</tr>
<tr>
<td>2</td>
<td>2045</td>
<td>23846</td>
<td>12957.83</td>
</tr>
<tr>
<td>1</td>
<td>74</td>
<td>9572</td>
<td>8492.05</td>
</tr>
<tr>
<td>634</td>
<td>175</td>
<td>12753</td>
<td>6876.11</td>
</tr>
<tr>
<td>361</td>
<td>375</td>
<td>3653</td>
<td>6422.93</td>
</tr>
</tbody>
</table>

(5 rows)
Solution for finding particular web requests:

Application adds comments to locate specific queries + auto_explain/log_min_duration_statement

/*
application:pganalyze,
controller:graphql,
action:graphql,
line:/app/services/dataload.rb:39:in `select_rows',
graphql:getQueryDetailStats,
request_id:55a6fa2d-9ffe-4374-a535-f5d1ee64ba84
*/
Built-in Wait Event Aggregation

pg_wait_sampling
Parallel Query

```
pg_stat_activity
(backend_type = parallel worker)
```

New
```
pg_stat_activity
(leader_pid)
```

New
```
EXPLAIN improvements
```
pg_stat_activity: leader_pid for Parallel Query

<table>
<thead>
<tr>
<th>backend_type</th>
<th>leader_pid</th>
<th>state</th>
<th>wait_event</th>
<th>wait_event_type</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>client backend</td>
<td>36936</td>
<td>active</td>
<td>DataFileRead</td>
<td>IO</td>
<td>SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;</td>
</tr>
<tr>
<td>parallel worker</td>
<td>36936</td>
<td>active</td>
<td>DataFileRead</td>
<td>IO</td>
<td>SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;</td>
</tr>
<tr>
<td>parallel worker</td>
<td>36936</td>
<td>active</td>
<td>DataFileRead</td>
<td>IO</td>
<td>SELECT * FROM log_lines_30d ORDER BY occurred_at DESC LIMIT 10;</td>
</tr>
</tbody>
</table>

(3 rows)
New in Postgres 13

EXPLAIN improvements for parallel workers

QUERY PLAN

Limit (cost=2333874.04..2333876.38 rows=20 width=206) (actual time=29049.924..29049.934 rows=20 loops=1)
  ->  Gather Merge (cost=2333874.04..7549291.60 rows=44700458 width=206) (actual time=29046.525..29217.937 rows=20 loops=1)
      Workers Planned: 2
      Workers Launched: 2
      ->  Sort (cost=2332874.02..2388749.59 rows=22350229 width=206) (actual time=28998.140..28998.143 rows=20 loops=3)
           Sort Key: log_lines_30d.occurred_at DESC
           Sort Method: top-N heapsort  Memory: 34kB
           Worker 0: Sort Method: top-N heapsort  Memory: 35kB
           Worker 1: Sort Method: top-N heapsort  Memory: 35kB
      ...

+ JIT Information
+ JSON format fixes
Aggregate information about Effectiveness of Parallel Query

Are my queries using parallel query?
Are there sufficient workers for parallel query?
Query Failures

**Log Events**

- Canceling statement due to statement timeout
- Canceling statement due to user request

*New* CONTEXT for failure of parameterized queries
CONTEXT for failure of parameterized queries

ERROR: division by zero
STATEMENT: SELECT 1/$1

SET log_parameter_max_length_on_error = 1024

ERROR: division by zero
CONTEXT: extended query with parameters: $1 = '0'
STATEMENT: SELECT 1/$1
Heavyweight Locks

pg_locks

Log Events

Process acquired lock on tuple / relation / object
Process still waiting for lock on tuple / relation / object
Canceling statement due to lock timeout
Deadlock detected (transaction rolled back)
Process avoided deadlock by rearranging queue order
Aggregate Lock Statistics

Difficult to use `pg_locks` for historic data

(e.g. `pg_stat_statements` `lock_wait_time` column)
Table/Index access

- `pg_stat_all_tables`
- `pg_statio_all_tables`
- `pg_stat_all_indexes`
- `pg_statio_all_indexes`
Per-statement index scan/seq scan scan counters

pg_stat_statements should have idx_scan and seq_scan counters
CPU, I/O and Memory

System metrics

pg_statio_*.  
New pg_shmem_allocations
What’s Missing

Connection memory usage statistics
New in Postgres 13

pg_shmem_allocations

```sql
=# SELECT * FROM pg_shmem_allocations ORDER BY allocated_size DESC;

<table>
<thead>
<tr>
<th>name</th>
<th>off</th>
<th>size</th>
<th>allocated_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Blocks</td>
<td>86739584</td>
<td>8589934592</td>
<td>8589934592</td>
</tr>
<tr>
<td>&lt;anonymous&gt;</td>
<td></td>
<td>91191424</td>
<td>91191424</td>
</tr>
<tr>
<td>Buffer Descriptors</td>
<td>19630720</td>
<td>67108864</td>
<td>67108864</td>
</tr>
<tr>
<td>Buffer IO Locks</td>
<td>8676674176</td>
<td>33554432</td>
<td>33554432</td>
</tr>
<tr>
<td>Checkpoint Data</td>
<td>8808573696</td>
<td>25165888</td>
<td>25165952</td>
</tr>
<tr>
<td>Checkpoint BufferIds</td>
<td>8710228608</td>
<td>20971520</td>
<td>20971520</td>
</tr>
<tr>
<td>XLOG Ctl</td>
<td>104832</td>
<td>16803472</td>
<td>16803584</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8033024</td>
<td>8033024</td>
</tr>
<tr>
<td>Xact</td>
<td>16908800</td>
<td>2116320</td>
<td>2116352</td>
</tr>
<tr>
<td>Backend Activity Buffer</td>
<td>8807698304</td>
<td>541696</td>
<td>541696</td>
</tr>
<tr>
<td>Subtrans</td>
<td>19158912</td>
<td>267008</td>
<td>267008</td>
</tr>
<tr>
<td>Backend Status Array</td>
<td>8807406080</td>
<td>224396</td>
<td>224396</td>
</tr>
</tbody>
</table>
```

<= shared_buffers
WAL Writing

pg_current_wal_lsn

New Per-statement WAL statistics
New autovacuum WAL statistics
New EXPLAIN WAL statistics
New in Postgres 13

Per-statement WAL statistics

=# SELECT substring(query for 70), wal_records, wal_fpi, wal_bytes FROM pg_stat_statements ORDER BY wal_records DESC;

<table>
<thead>
<tr>
<th>substring</th>
<th>wal_records</th>
<th>wal_fpi</th>
<th>wal_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY public.log_lines_30d_20200525 (log_line_id, server_id, log_line_p)</td>
<td>13637990</td>
<td>1690272</td>
<td>11920182549</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, backend_i)</td>
<td>8568987</td>
<td>5429</td>
<td>881673525</td>
</tr>
<tr>
<td>COPY activity.query_origins_7d_20200525 (backend_query_id, database_id)</td>
<td>7507811</td>
<td>789923</td>
<td>5796656931</td>
</tr>
<tr>
<td>COPY activity.backend_snapshots_1d_20200525 (collected_at, state, wait)</td>
<td>6909068</td>
<td>802625</td>
<td>5241077274</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, identity)</td>
<td>6541995</td>
<td>11087</td>
<td>705700102</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLE upsert_data (server_id uuid NOT NULL, identity)</td>
<td>6418566</td>
<td>31123</td>
<td>771654086</td>
</tr>
<tr>
<td>COPY public.log_line_stats_30d_20200525 (log_line_id, server_id, occur)</td>
<td>5338310</td>
<td>722396</td>
<td>4800621926</td>
</tr>
<tr>
<td>DROP TABLE upsert_data</td>
<td>5056385</td>
<td>13</td>
<td>269723603</td>
</tr>
<tr>
<td>COPY public.log_lines_30d_20200524 (log_line_id, server_id, log_line_p)</td>
<td>3162220</td>
<td>496058</td>
<td>3261770520</td>
</tr>
<tr>
<td>DROP TABLE upsert_data</td>
<td>2134608</td>
<td>6</td>
<td>113901008</td>
</tr>
</tbody>
</table>
...
New in Postgres 13

autovacuum WAL statistics

LOG: automatic vacuum of table “…”: index scans: 1
pages: 0 removed, 75444 remain, 3 skipped due to pins, 0 skipped frozen
tuples: 996760 removed, 4210912 remain, 0 are dead but not yet removable, oldest xmin: 1871789
buffer usage: 114171 hits, 1 misses, 21614 dirtied
avg read rate: 0.001 MB/s, avg write rate: 20.434 MB/s
system usage: CPU: user: 2.42 s, system: 0.03 s, elapsed: 8.26 s
WAL usage: 94064 records, 17930 full page images, 34394711 bytes
**New in Postgres 13**

**EXPLAIN WAL statistics**

```sql
=# BEGIN;
BEGIN
=*
=# EXPLAIN (ANALYZE, WAL) UPDATE backend_counts SET state = state WHERE backend_count_id IN (SELECT backend_count_id FROM backend_counts LIMIT 100);
QUERY PLAN

Update on backend_counts  (cost=4.47..850.04 rows=100 width=139) (actual time=1.049..1.049 rows=0 loops=1)
  WAL: records=168 fpi=5 bytes=39013
    ->  Nested Loop  (cost=4.47..850.04 rows=100 width=139) (actual time=0.239..0.734 rows=100 loops=1)
      WAL: records=2 bytes=416
        ->  HashAggregate  (cost=4.04..5.04 rows=100 width=56) (actual time=0.229..0.246 rows=100 loops=1)
          Group Key: "ANY_subquery".backend_count_id
          Peak Memory Usage: 45 kB
          WAL: records=2 bytes=416
            ->  Subquery Scan on "ANY_subquery"  (cost=0.00..3.79 rows=100 width=56) (actual time=0.014..0.202 rows=100 loops=1)
              WAL: records=168 fpi=5 bytes=39013
              ->  Limit  (cost=0.00..2.79 rows=100 width=16) (actual time=0.011..0.184 rows=100 loops=1)
                ->  Seq Scan on backend_counts backend_counts_1  (cost=0.00..119801.53 rows=4291453 width=16) (actual time=0.010..0.177 rows=100)
                  Index Cond: (backend_count_id = "ANY_subquery".backend_count_id)
                  WAL: records=2 bytes=416
```
Utility Commands

pg_stat_progress_vacuum

New pg_stat_progress_analyze
pg_stat_progress_cluster
pg_stat_progress_create_index
### pg_stat_progress_analyze

```
=## SELECT * FROM pg_stat_progress_analyze ;

   pid  | datid |      datname      | relic  |         phase         | sample_blks_total | sample_blks_scanned | ext_stats_total
-------+-------+-------------------+--------+-----------------------+-------------------+---------------------+---------------
  36936 | 16400 | pganalyze_staging | 115537 | acquiring sample rows |           30000   |                26756 |               0

(1 row)
```
autovacuum

pg_stat_progress_vacuum

**Log Events**

- Canceling autovacuum task
- Database must be vacuumed within N transactions (TXID Wraparound Warning)
- Database is not accepting commands to avoid wraparound data loss
- Autovacuum launcher started
- Autovacuum launcher shutting down
- Automatic vacuum of table completed
- Skipping vacuum - lock not available
Aggregate autovacuum stats (only available in logs)

How often a table is being vacuumed
Avg runtime of a vacuum
# Tuples that couldn’t be removed
Backups

New  pg_stat_progress_basebackup
New in Postgres 13

pg_stat_progress_basebackup

=\# SELECT * FROM pg_stat_progress_basebackup ;

<table>
<thead>
<tr>
<th>pid</th>
<th>phase</th>
<th>backup_total</th>
<th>backup_streamed</th>
<th>tablespaces_total</th>
<th>tablespaces_streamed</th>
</tr>
</thead>
<tbody>
<tr>
<td>35397</td>
<td>waiting for checkpoint to finish</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(1 row)

=\# SELECT *, backup_streamed / backup_total::float * 100 AS pct_done FROM pg_stat_progress_basebackup ;

<table>
<thead>
<tr>
<th>pid</th>
<th>phase</th>
<th>backup_total</th>
<th>backup_streamed</th>
<th>tablespaces_total</th>
<th>tablespaces_streamed</th>
<th>pct_done</th>
</tr>
</thead>
<tbody>
<tr>
<td>35397</td>
<td>streaming database files</td>
<td>63006018048</td>
<td>52671390720</td>
<td>1</td>
<td>0</td>
<td>83.59739648976586</td>
</tr>
</tbody>
</table>

(1 row)
What’s Missing

1. Client-side connection latency
2. Aggregation of security-relevant Postgres events
3. Aggregate Plan Statistics
4. Breakdown of non-waiting active state
5. Query Progress Monitoring
6. pgss: Better handling of IN(...) lists & other ORM patterns
7. Linking pg_stat_statements with other views & logs
8. Finding queries based on application requests/customers
9. Built-in Wait Event Aggregation
10. Aggregate information about effectiveness of Parallel Query
11. Aggregate Lock Statistics
12. Per-statement index scan/seq scan counters
13. Connection memory usage statistics
14. Aggregate autovacuum stats
New in Postgres 13

1. EXPLAIN: Buffers for Planning
2. pg_stat_statements: Planning Time
3. Additional & renamed wait events
4. pg_stat_activity: leader.pid for Parallel Query
5. EXPLAIN improvements for parallel workers
6. CONTEXT for failure of parameterized queries
7. pg_shmem_allocations
8. Per-statement WAL statistics
9. autovacuum WAL statistics
10. EXPLAIN WAL statistics
11. pg_stat_progress_analyze
12. pg_stat_progress_basebackup
Thank you!

lukas@fittl.com

@LukasFittl