

# Identifying Slow Queries, and Fixing Them!

Stephen Frost  
Crunchy Data  
[stephen@crunchydata.com](mailto:stephen@crunchydata.com)

PGCon 2020  
May 27, 2020



# Stephen Frost

- Chief Technology Officer @ Crunchy Data
- Committer, PostgreSQL
- Major Contributor, PostgreSQL
- GSSAPI Encryption in v12
- Row-Level Security in 9.5
- Column-level privileges in 8.4
- Implemented the roles system in 8.1
- Contributions to PL/pgSQL, PostGIS

# Community!

- Follow Planet PostgreSQL! <https://planet.postgresql.org>
- Join PostgreSQL.EU! <https://postgresql.eu>
- Join PostgreSQL.US! <https://postgresql.us>

# Finding Slow Queries

- Logging- Enable with postgresql.conf
- Log Analysis- Generate reports (pgBadger)
- Viewing Active Queries (pg\_stat\_statements)

# Logging

postgresql.conf configuration

- log\_min\_duration\_statement
- log\_line\_prefix
- log\_checkpoints
- log\_connections
- log\_disconnections
- log\_lock\_waits
- log\_temp\_files
- log\_autovacuum\_min\_duration

# log\_min\_duration\_statement

```
log_min_duration_statement = 0
```

- Zero Logs every statement sent
- Number is in milliseconds
- Queries taking longer than value logged
- Includes duration \*on the same line\*
- Do NOT enable log\_statement or log\_duration

Result:

```
LOG:  duration: 1001.474 ms  statement: select pg_sleep(1);
```

# log\_line\_prefix

Pre-pended to every log message.

```
log_line_prefix = '%t [%p]: [%l-1] %quser=%u,db=%d,app=%a,client=%h '
```

Includes:

- %t - Timestamp
- %p - Process ID (pid)
- %l - Session Line Number
- %u - Logged in user
- %d - Database logged in to
- %a - Application name (if set)
- %h - Remote host
- %q - Stop here in non-session processes

Result:

```
2016-09-12 14:43:04 EDT [2830]: [11-1] ...  
    user=sfrost,db=postgres,app=psql,client=[local] ...  
LOG:  duration: 1001.193 ms  statement: select pg_sleep(1);
```

# log\_checkpoints

Logs information about each checkpoint

```
log_checkpoints = on
```

Includes:

- When/Why the checkpoint started
- When the checkpoint completed
- Statistics regarding what happened during checkpoint

Result:

```
2016-09-12 14:51:02 EDT [2609]: [3-1] LOG: ...
checkpoint starting: immediate force wait
2016-09-12 14:51:02 EDT [2609]: [4-1] LOG: ...
checkpoint complete: wrote 67 buffers (0.4%); ...
0 transaction log file(s) added, 0 removed, 0 recycled; ...
write=0.000 s, sync=0.059 s, total=0.068 s; sync files=18, ...
longest=0.025 s, average=0.003 s; distance=88 kB, estimate=88 kB
```



# Connection logging

Logs information about each connection and disconnection

```
log_connection = on  
log_disconnection = on
```

Includes:

- When/Why the checkpoint started
- When the checkpoint completed
- Statistics regarding what happened during checkpoint

Result:

```
2016-09-12 15:07:07 EDT [19608]: [1-1] user=[unknown],db=[unknown],...  
  app=[unknown],client=[local] LOG:  connection received: host=[local]  
2016-09-12 15:07:07 EDT [19608]: [2-1] user=sfrost,db=postgres,...  
  app=[unknown],client=[local] ...  
  LOG:  connection authorized: user=sfrost database=postgres  
2016-09-12 15:07:08 EDT [19608]: [3-1] user=sfrost,db=postgres,...  
  app=psql,client=[local] LOG:  disconnection: ...  
  session time: 0:00:01.231 user=sfrost database=postgres host=[local]
```

# log\_lock\_waits

Logs information when a query waits on a lock

```
log_lock_waits = on
```

Fires after 1s (deadlock\_timeout). Result:

```
2016-09-12 16:44:14 EDT [29554]: [8-1] user=sfrost,db=postgres,...
  app=psql,client=[local] LOG:  process 29554 ...
  still waiting for ShareLock on transaction 668 after 1000.045 ms
2016-09-12 16:44:14 EDT [29554]: [9-1] user=sfrost,db=postgres,...
  app=psql,client=[local] DETAIL:  ...
  Process holding the lock: 29617. Wait queue: 29554.
2016-09-12 16:44:14 EDT [29554]: [10-1] user=sfrost,db=postgres,...
  app=psql,client=[local] CONTEXT:  ...
  while locking tuple (0,1) in relation "t1"
2016-09-12 16:44:14 EDT [29554]: [11-1] user=sfrost,db=postgres,...
  app=psql,client=[local] STATEMENT:  select * from t1 for update;
```

# log\_temp\_files

Logs information when a query needs to create temp files

```
log_temp_files = 0
```

Value is how large the temp file is, zero means all. Result:

```
2016-09-12 17:06:04 EDT [29554]: [51-1] user=sfrost,db=postgres,...
    app=psql,client=[local] LOG: ...
    temporary file: path "base/pgsql_tmp/pgsql_tmp29554.2", size 1540096
2016-09-12 17:06:04 EDT [29554]: [52-1] user=sfrost,db=postgres,...
    app=psql,client=[local] STATEMENT:  select * from t1 order by 1;
```

# log\_autovacuum\_min\_duration

Logs autovacuum activity

```
log_autovacuum_min_duration = 0
```

Value is how long the autovacuum command took

```
2016-09-12 17:10:56 EDT [357]: [1-1] LOG: ...
    automatic vacuum of table "postgres.public.t1": index scans: 0
        pages: 487 removed, 0 remain, 0 skipped due to pins
        tuples: 110000 removed, 0 remain, 0 are dead but not yet removable
        buffer usage: 1480 hits, 2 misses, 3 dirtied
        avg read rate: 0.107 MB/s, avg write rate: 0.160 MB/s
        system usage: CPU 0.00s/0.02u sec elapsed 0.14 sec
2016-09-12 17:10:56 EDT [357]: [2-1] LOG: ...
    automatic analyze of table "postgres.public.t1" ...
    system usage: CPU 0.00s/0.00u sec elapsed 0.00 sec
```

# Log Analysis

## Running pgBadger

- apt-get install pgbadger
- pgbadger logfile
- Fancy reports!

# pg\_stat\_statements

## Installing pg\_stat\_statements

```
shared_preload_libraries = 'pg_stat_statements'  
track_io_timing = on
```

- Restart (not reload) PostgreSQL

```
sfrost@beorn:~# psql  
psql (12.1 (Ubuntu 12.1-1.pgdg19.04+1))  
=# create extension pg_stat_statements;
```

# pg\_stat\_statements

## Reviewing pg\_stat\_statements

View "public.pg\_stat\_statements"

Column	Type	Modifiers
userid	oid	
dbid	oid	
queryid	bigint	
query	text	
calls	bigint	
total_time	double precision	
min_time	double precision	
max_time	double precision	
mean_time	double precision	
stddev_time	double precision	
rows	bigint	
...		

# pg\_stat\_statements

## Reviewing pg\_stat\_statements

View "public.pg\_stat\_statements"

Column	Type	Modifiers
shared_blks_hit	bigint	
shared_blks_read	bigint	
shared_blks_dirtied	bigint	
shared_blks_written	bigint	
local_blks_hit	bigint	
local_blks_read	bigint	
local_blks_dirtied	bigint	
local_blks_written	bigint	
temp_blks_read	bigint	
temp_blks_written	bigint	
blk_read_time	double precision	
blk_write_time	double precision	



# pg\_stat\_statements

## Reviewing pg\_stat\_statements

```
queryid          | 3374102836  
query            | UPDATE pgbench_tellers  
                | SET tbalance = tbalance + ? WHERE tid = ?;  
calls           | 40000  
total_time      | 4735.070000000014  
min_time        | 0.012  
max_time        | 142.15  
mean_time       | 0.11837675  
stddev_time     | 1.30052157525719  
rows            | 40000
```

# pg\_stat\_statements

## Reviewing pg\_stat\_statements

```
queryid          | 3619888345  
query            | SELECT abalance FROM pgbench_accounts WHERE aid = ?;  
calls           | 40000  
total_time      | 516.500999999987  
min_time        | 0.008  
max_time        | 0.085  
mean_time       | 0.0129125249999999  
stddev_time     | 0.00338086869374945  
rows            | 40000
```

# Understanding Why Queries Are Slow

- PostgreSQL Configuration Issues
- Dead tuples / bloat
- Query Plan

# PostgreSQL Configuration

- work\_mem
- maintenance\_work\_mem
- effective\_cache\_size
- shared\_buffers
- checkpoint\_segments
- min\_wal\_size
- max\_wal\_size
- checkpoint\_timeout
- checkpoint\_completion\_target

# PostgreSQL Configuration - work\_mem

- May be allocated many times over
- Also used for bitmaps max size; bitmaps reduce their accuracy when its too much.

# PostgreSQL Configuration - maintenance\_work\_mem

- Allocated by autovacuum worker process, as needed
- All parallel CREATE INDEX processes will only use up to maintenance\_work\_mem in total

# PostgreSQL Configuration - `effective_cache_size`

- NEVER actually allocated
- Estimate of size of disk cache
- Larger increases index usage, might not always be helpful

# PostgreSQL Configuration - shared\_buffers

- Allocated at server start
- Caches disk pages, more-or-less exactly
- 25 - 50 percent of system memory is typical
- pg\_buffercache useful to analyze contents



# PostgreSQL Configuration - checkpoints, wal\_size

- `min_wal_size`
  - Minimum size of the WAL to maintain
  - Creating new WAL files is not free
- `max_wal_size`
  - Maximum size of WAL to allow
  - If too low, checkpoints will happen BEFORE checkpoint timeout!
- `checkpoint_segments`
  - Old option, replaced by `max_wal_size`
- `checkpoint_timeout`
  - Controls length of time between checkpoints
  - WAL replay starts from last checkpoint on crash
- `checkpoint_completion_target`
  - How much of checkpoint timeout to use to perform a checkpoint

# Dead Tuples / Bloat

- VACUUM marks records as reusable
  - Reusable tuples used for new inserts/updates
  - PG still has to consider those tuples in scans, etc
- Bloat
  - Table can have lots of dead tuples
  - Indexes can have bloat also
- check\_postgres.pl
  - Helps identify tables to check for bloat
  - Some bloat is helpful
- Eliminating all bloat requires a rewrite
- CLUSTER / VACUUM FULL

# Retrieving Data

- Sequentially step through EVERY record
  - Seq Scan Node
  - Bulk operation
  - Bitmap scan
- Use an index, pull SPECIFIC records
  - Index Scan Node
  - Indexes generally have to be created
  - Often requires accessing index and heap
  - Data can be returned in order
- Index Only Scan
  - Index Only Scan Node
  - Columns must be in index
  - May require going to the heap
  - VACUUM updates visibility map

# Putting things together (Joins)

- Nested Loop
  - Step through one table
  - For each step, look up record in other table
  - Fast- for small sets, not good for bulk
- Merge Join
  - Order (sort) each table
  - Walk through both tables, return matches
  - Good for bulk operations
  - Sorting is expensive, can use index
- Hash Join
  - Scan one table and build a hash table
  - Step through other table using the hash table to find matches
  - Slow start
  - Very fast, but memory intensive

# Adding it all up (Aggregates)

- Group Agg
  - Order / sort input
  - Step through each record, if it matches last, combine
  - Sorting is expensive
- Hash Agg
  - Scan table, building hash table
  - Hash table matches are combined
  - Memory intensive

# What's the best plan?

- It Depends!
- Database gathers and uses statistics
  - ANALYZE
  - VACUUM ANALYZE
  - pg\_statistic
  - Autovacuum
- Bad stats = Bad plans
  - EXPLAIN ANALYZE
  - Check results vs. estimates
  - Statistics target

# Automating collection of plans

- auto\_explain
  - Logs explain for queries
  - Based on length of time
- Enabling

```
shared_preload_libraries = 'auto_explain'  
explain.log_min_duration = 50;  
explain.log_nested_statements = true;
```

- Can also do 'explain analyze', but very expensive!
- Logging nested statements

# Analyzing plans

- Explain output options
  - XML
  - JSON
  - YAML
- Tools for analyzing explain
  - pgAdmin3/4
  - [explain.depesz.com](http://explain.depesz.com)



# Fixing Slow Queries

## Low-hanging fruit

- Indexes
  - Seq Scan?
  - Only one row returned?
  - No aggregation?
  - Create an index
- work\_mem
  - Small data set?
  - Sorting happening?
  - Merge Join used?
  - Increase work\_mem

# Fixing Slow Queries

- Statistics
  - Large data set?
  - Nested Loop?
  - Ensure current statistics (ANALYZE)
  - Increase statistics target
- Indexes w/ Foreign Keys
  - DELETE is slow?
  - Table referred to with foreign key?
  - Create index on referring table

# Prepared Queries

- Plan Once, run many
  - Avoids repeated planning cost
  - Plan Cache has generic and specific plans
  - 5-time rule
- Explain analyze with execute

```
prepare q as select * from mytable where x = $1;  
explain execute q('myid');  
explain analyze execute q('myid');
```

# Query Review

- `select count(*) from table;`
  - Index can help- Index Only Scan
  - Still must check all records
- `select * from table;`
  - Returns all columns and rows...
  - Is every row needed?
  - Is every column needed?
  - de-TOAST can be expensive
- `select distinct * from a, b, c where a.x = b.x;`
  - Watch out for 'select distinct'
  - Missing join condition for 'c'
  - Cartesian product created, then dups removed
  - Join syntax is better
  - `select * from a join b using (x) join c using (x);`

# More Queries

- `select * from x where myid in (select myid from bigtable);`
  - Could be turned into a join
  - Joins allow more options for how to execute the query
  - Generally, a faster way is found
- `select * from x where myid not in (select myid from bigtable);`
  - Left-join can be used instead
  - May be able to use `NOT EXISTS` instead

# Even More Queries

- Use CTEs
  - Keep the results of them small
  - WITH cte AS (select \* from expensive join)
  - select cte.result, othertable.x from cte join othertable;
- Really, really faster count(\*) estimate
  - Use the database statistics
  - pg\_class.reltuples
  - Only useful for whole tables
  - Will not be perfect
  - Trigger-based approach

# Review, and then some

- Tuning PG
  - Increase `work_mem`, `maintenance_work_mem`
  - Set `effective_cache_size` based on memory
  - Increase `shared_buffers`
- Partial Indexes / Functional Indexes
  - Index only part of the table
  - Use a function inside an index
  - Double-check query plans use the index
- Remove unused indexes
  - Unused indexes still have to be maintained
  - More indexes, slower writes
  - PG statistics- review `pg_stat_user_indexes`

# Questions?

- Questions?
- Follow Planet PostgreSQL! <https://planet.postgresql.org>
- Join PostgreSQL.EU! <https://postgresql.eu>
- Join PostgreSQL.US! <https://postgresql.us>

Thanks!