

pganalyze

Stack-based Instrumentation

Presented by Lukas Fittl

Agenda for today

- **EXPLAIN ANALYZE** was slow, and now its faster
- **BUFFERS** is still a bottleneck
- How **Stack-Based Instrumentation** Works
- Splitting Out **Index Scan Table Access**
- Other Related Work

EXPLAIN ANALYZE

was slow,

and now its faster

EXPLAIN ANALYZE with less timing overhead

pganalyze

EXPLAIN (ANALYZE, TIMING) is not free - back in 2020:

```
SELECT count(*) FROM lotsarows;

-- best of three:
SELECT count(*) FROM lotsarows;
Time: 1923.394 ms (00:01.923)

-- best of three:
EXPLAIN (ANALYZE, TIMING OFF, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 2319.830 ms (00:02.320)

-- best of three:
EXPLAIN (ANALYZE, TIMING ON, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 4202.649 ms (00:04.203)
```

118% overhead!

EXPLAIN ANALYZE with less timing overhead

pganalyze

EXPLAIN (ANALYZE, TIMING) is not free - perf profile shows why:

```
-   95.49%     0.00%  postgres     postgres                  [.] agg_retrieve_direct
(inlined)
   - agg_retrieve_direct (inlined)
      - 79.27% fetch_input_tuple
         - ExecProcNode (inlined)
            - 75.72% ExecProcNodeInstr
               + 25.22% SeqNext
               - 21.74% InstrStopNode
                  + 17.80% __GI___clock_gettime (inlined)
               - 21.44% InstrStartNode
                  + 19.23% __GI___clock_gettime (inlined)
               + 4.06% ExecScan
      + 13.09% advance_aggregates (inlined)
        1.06% MemoryContextReset
```

EXPLAIN ANALYZE with less timing overhead

pganalyze

Postgres 19 now uses RDTSC instead of clock_gettime() for timing information

```
SELECT count(*) FROM lotsarows;

-- best of three:
SELECT count(*) FROM lotsarows;
Time: 268.466 ms

-- best of three:
EXPLAIN (ANALYZE, TIMING OFF, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 297.043 ms

-- best of three:
EXPLAIN (ANALYZE, TIMING ON, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 351.773 ms
```

Before: 118%

Now: 30% overhead

EXPLAIN ANALYZE with less timing overhead

pganalyze

Controllable through timing_clock_source

```
SET timing_clock_source = 'tsc';

-- best of three:
EXPLAIN (ANALYZE, TIMING ON, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 355.690 ms

SET timing_clock_source = 'system';

-- best of three:
EXPLAIN (ANALYZE, TIMING ON, BUFFERS OFF) SELECT count(*) FROM lotsarows;
Time: 801.979 ms
```

EXPLAIN ANALYZE with less timing overhead

pganalyze

pg_test_timing shows different clock source speeds:

```
System clock source: clock_gettime (CLOCK_MONOTONIC)
Average loop time including overhead: 18.80 ns
Histogram of timing durations:
...

Clock source: RDTSCP
Average loop time including overhead: 16.94 ns
...

Fast clock source: RDTSC
Average loop time including overhead: 11.69 ns
...

TSC frequency in use: 2993629 kHz
TSC frequency from calibration: 2994357 kHz
TSC clock source will be used by default, unless
timing_clock_source is set to 'system'.
```

EXPLAIN ANALYZE with less timing overhead

pganalyze

We also moved some instrumentation code in 19, which helped further:

instrumentation: Move ExecProcNodeInstr to allow inlining

```
author    Andres Freund <andres@anarazel.de>
          Wed, 8 Apr 2026 01:36:07 +0000 (21:36 -0400)
committer Andres Freund <andres@anarazel.de>
          Wed, 8 Apr 2026 01:36:49 +0000 (21:36 -0400)
commit    544000288ec8f7dc6a1e0285821adc47324ecd33
tree      ec84c83e2b6d293fa7e0184215b34cb593964678   tree
parent    e157fe6f76e0be4f05a725b3099cf53949a2c2af   commit | diff
```

```
instrumentation: Move ExecProcNodeInstr to allow inlining

This moves the implementation of ExecProcNodeInstr, the ExecProcNode variant
that gets used when instrumentation is on, to be defined in instrument.c
instead of execProcNode.c, and marks functions it uses as inline.

This allows compilers to generate an optimized implementation, and shows a 4
to 12% reduction in instrumentation overhead for queries that move lots of
rows.

Author: Lukas Fittl <lukas@fittl.com>
Suggested-by: Andres Freund <andres@anarazel.de>
Reviewed-by: Andres Freund <andres@anarazel.de>
Discussion: https://postgr.es/m/CAP53PkzdBK8VJ1fS4AZ481LgMN8f9mJiC39ZRHqkFUSYq6KWmg@mail.gmail.com
```

```
src/backend/executor/execProcnode.c   diff | blob | history
src/backend/executor/instrument.c     diff | blob | history
src/include/executor/executor.h       diff | blob | history
```

4 to 12% performance improvement for some workloads

BUFFERS

is still a bottleneck

BUFFERS is still a bottleneck

pganalyze

Benchmark:

```
CREATE TABLE test(id int);
INSERT INTO test SELECT * FROM generate_series(0, 10000000);
SET max_parallel_workers_per_gather = 0;

SELECT COUNT(*) FROM test;
Time: 233.261ms (best of three)

EXPLAIN (ANALYZE, BUFFERS ON, TIMING ON) SELECT COUNT(*) FROM test;
Time: 337.335ms (best of three)

EXPLAIN (ANALYZE, BUFFERS ON, TIMING OFF) SELECT COUNT(*) FROM test;
Time: 289.394ms (best of three)

EXPLAIN (ANALYZE, BUFFERS OFF, TIMING OFF) SELECT COUNT(*) FROM test;
Time: 250.436ms (best of three)
```

~24% overhead for BUFFERS

~7% overhead for rows only

BUFFERS is still a bottleneck

pganalyze

Buffer Usage counting is slow because of **diffing** in InstrStartNode/InstrStopNode and BufferUsageAccumDiff**.**

Can we drop this? **Yes we can.**

```
- /* dst += add - sub */
- void
- BufferUsageAccumDiff(BufferUsage *dst,
-                                     const BufferUsage *add,
-                                     const BufferUsage *sub)
- {
-         dst->shared_blks_hit += add->shared_blks_hit - sub->shared_blks_hit;
-         dst->shared_blks_read += add->shared_blks_read - sub->shared_blks_read;
-         dst->shared_blks_dirtied += add->shared_blks_dirtied - sub->shared_blks_dirtied;
-         dst->shared_blks_written += add->shared_blks_written - sub->shared_blks_written;
-         dst->local_blks_hit += add->local_blks_hit - sub->local_blks_hit;
-         dst->local_blks_read += add->local_blks_read - sub->local_blks_read;
-         dst->local_blks_dirtied += add->local_blks_dirtied - sub->local_blks_dirtied;
-         dst->local_blks_written += add->local_blks_written - sub->local_blks_written;
-         dst->temp_blks_read += add->temp_blks_read - sub->temp_blks_read;
-         dst->temp_blks_written += add->temp_blks_written - sub->temp_blks_written;
-         INSTR_TIME_ACCUM_DIFF(dst->shared_blk_read_time,
-                               add->shared_blk_read_time, sub->shared_blk_read_time);
-         INSTR_TIME_ACCUM_DIFF(dst->shared_blk_write_time,
```

BUFFERS is still a bottleneck - not anymore!

pganalyze

Benchmark before:

```
SELECT COUNT(*) FROM test;
Time: 233.261ms (best of three)

EXPLAIN (ANALYZE, BUFFERS ON, TIMING OFF) SELECT COUNT(*) FROM test;
Time: 289.394ms (best of three)
```

Benchmark with patch:

```
SELECT COUNT(*) FROM test;
Time: 234.201ms (best of three)

EXPLAIN (ANALYZE, BUFFERS ON, TIMING OFF) SELECT COUNT(*) FROM test;
Time: 252.485ms (best of three)
```

~7% overhead for BUFFERS

How **Stack-Based Instrumentation** Works

How Stack-Based Instrumentation Works

Before - In Each Repeated Activity:

InstrStartNode - remember **pgBufferUsage**



InstrStopNode - take current **pgBufferUsage** and call **BufferUsageAccumDiff**

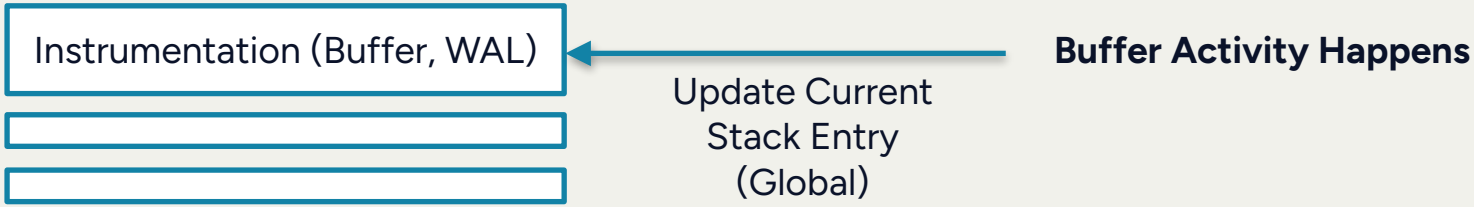
At Query End:

No extra work, per-node instrumentation already done.

How Stack-Based Instrumentation Works

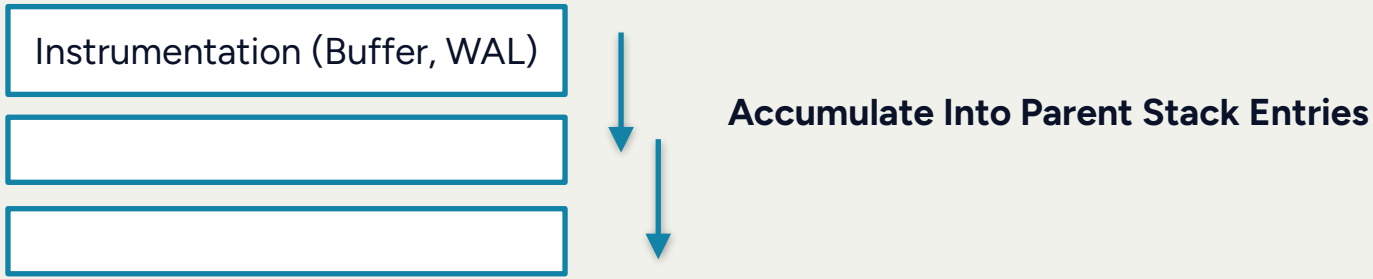
After - In Each Repeated Activity:

InstrStartNode - push **per-node Instrumentation on stack:**



InstrStopNode - pop back to **parent stack entry**

At Query End:



Splitting Out **Index Scan Table Access**

Splitting Out Index Scan Table Access

pganalyze

Example:

```
CREATE TABLE test2(id int, id2 int);
INSERT INTO test SELECT x, x*2 FROM generate_series(0, 10000000) _(x);
SET max_parallel_workers_per_gather = 0;

EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM test2 WHERE id2 IN (123, 456) AND id = 10;

 Index Scan using test2_id2_idx on public.test2  (...)
   Output: id, id2
   Index Cond: (test2.id2 = ANY ('{123,456}'::integer[]))
   Filter: (test2.id = 10)
   Rows Removed by Filter: 1
   Index Searches: 1
   Table Buffers: shared read=1
   Buffers: shared hit=3 read=1
```

Other Related Work

Other Related Work

pganalyze

shared_blks_hit double counts - Setup:

```
CREATE TABLE a(id int);
CREATE TABLE b(id int, a_id int);

INSERT INTO a SELECT * FROM generate_series(0, 1_000_000);
INSERT INTO b SELECT a, a FROM generate_series(0, 1_000_000) x(a);

CREATE INDEX ON a(id);
CREATE INDEX ON b(id);
CREATE INDEX ON b(a_id);
VACUUM ANALYZE a;
VACUUM ANALYZE b;

SET max_parallel_workers_per_gather = 0;
```

Other Related Work

pganalyze

shared_blks_hit double counts - Warm cache:

```
EXPLAIN ANALYZE
SELECT * FROM a JOIN b ON (a.id = b.a_id)
WHERE b.id BETWEEN 500 AND 5000;

Nested Loop  (...)
   Buffers: shared hit=13540
   ->  Index Scan using b_id_idx on b  (...)
         Index Cond: ((id >= 500) AND (id <= 5000))
         Index Searches: 1
         Buffers: shared hit=36
   ->  Index Only Scan using a_id_idx on a  (...)
         Index Cond: (id = b.a_id)
         Heap Fetches: 0
         Index Searches: 4501
         Buffers: shared hit=13504
```

Other Related Work

pganalyze

shared_blks_hit double counts - After restart:

```
EXPLAIN ANALYZE
SELECT * FROM a JOIN b ON (a.id = b.a_id)
WHERE b.id BETWEEN 500 AND 5000;

Nested Loop  (...)
   Buffers: shared hit=13493 read=47
   ->  Index Scan using b_id_idx on b  (...)
         Index Cond: ((id >= 500) AND (id <= 5000))
         Index Searches: 1
         Buffers: shared hit=2 read=34
   ->  Index Only Scan using a_id_idx on a  (...)
         Index Cond: (id = b.a_id)
         Heap Fetches: 0
         Index Searches: 4501
         Buffers: shared hit=13491 read=13
```

Other Related Work

pganalyze

How to enable distinct buffer hit counts?

Storing all distinct buffer IDs would
be too expensive.

HyperLogLog (HLL) to the rescue!

Add an HLL structure to the stack, count distinct buffers when EXPLAIN (ANALYZE, BUFFERS DISTINCT) is on

Other Related Work

pganalyze

This works:

```
EXPLAIN (ANALYZE, BUFFERS DISTINCT)
SELECT * FROM a JOIN b ON (a.id = b.a_id)
WHERE b.id BETWEEN 500 AND 5000;

Nested Loop  (...)
   Buffers: shared hit=13540 hit distinct=0
   ->  Index Scan using b_id_idx on b  (...)
         Index Cond: ((id >= 500) AND (id <= 5000))
         Index Searches: 1
         Buffers: shared hit=36 hit distinct=36
   ->  Index Only Scan using a_id_idx on a  (....)
         Index Cond: (id = b.a_id)
         Heap Fetches: 0
         Index Searches: 4501
         Buffers: shared hit=13504 hit distinct=16
```

Estimate: 36 buffers
Actual: 36 buffers

Estimate: 16 buffers
Actual: 13 buffers

Other Related Work

pganalyze

Parallel Query Instrumentation Is Messy:

- We allocate shared memory for the worker instrumentation the whole lifecycle, but then only use it at the end (by copying from a local copy)

  What if we just updated the shared memory directly?
  (might also help with query progress reporting)

- We combine some metrics across leaders and workers, and keep others separate. With VERBOSE we show the workers split out, but not the leader.

  What if we merge at display time when doing EXPLAIN & make it consistent?

Thank you!



pganalyze

Learn more at pganalyze.com

Contact me: lukas@pganalyze.com