

Analyzing postgres performance problems using perf and eBPF

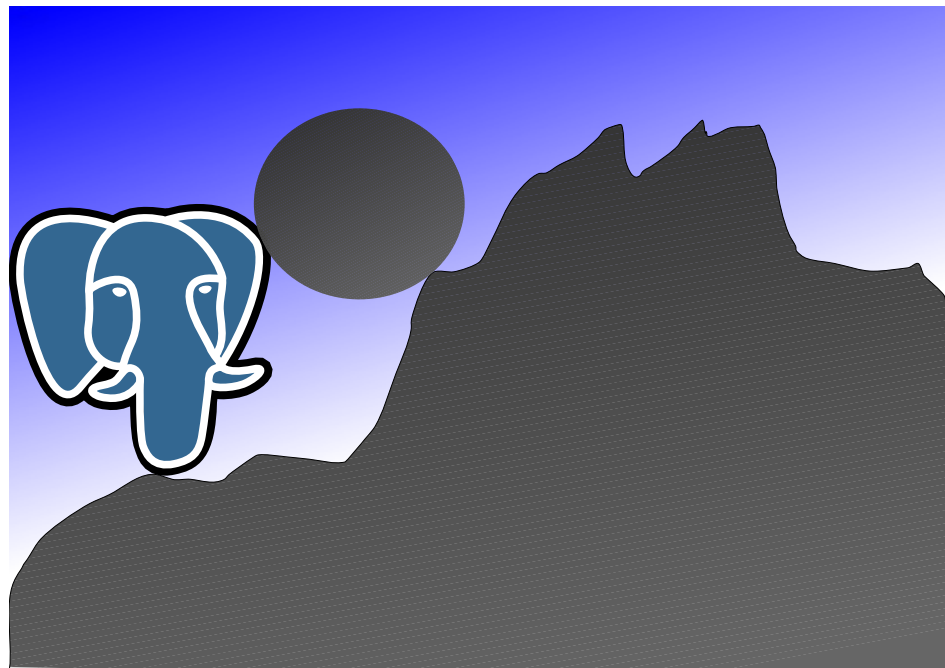
Andres Freund

PostgreSQL Developer,
Committer and Core Team
Member

andres@anarazel.de

andres.freund@microsoft.com

[@AndresFreundTec](https://twitter.com/AndresFreundTec)



<https://anarazel.de/talks/2022-04-12-cituscon/perf-bpf.pdf>

Perf

Perf

- Linux kernel profiling tool
- Requires elevated privileges
- “perf record” – profile workload
 - -p \$pid – one pid
 - -a – everyone
 - --call-graph dwarf – call graphs
- “perf report” – look at profile
 - --children / --no-children
 - --sort fielda,fieldb
- Brendan Gregg’s collection of perf links: <https://www.brendangregg.com/perf.html>

Initial Investigation

perf record -a sleep 3

perf report --sort comm,dso

Samples: 160K of event 'cycles', Event count (approx.): 118788343308

Overhead	Command	Shared Object
72.15%	<u>pgbench</u>	<u>[kernel.vmlinux]</u>
<u>10.69%</u>	postgres	postgres
4.76%	postgres	[kernel.vmlinux]
2.06%	pgbench	libcrypto.so.1.0.2k
1.32%	pgbench	libpthread-2.17.so
1.26%	pgbench	libc-2.17.so
1.25%	postgres	libcrypto.so.1.1
1.18%	swapper	[kernel.vmlinux]
0.99%	pgbench	libpq.so.5.14
0.91%	postgres	libc-2.33.so
0.90%	postgres	libssl.so.1.1
0.69%	pgbench	pgbench

perf record -p \$pgbench_pid sleep 1

perf report

Samples: 153K of event 'cycles', Event count (approx.): 91833386559

Overhead	Command	Shared Object	Symbol
69.90%	pgbench	<u>[kernel.vmlinux]</u>	<u>[k] <u>queued spin lock slowpath</u></u>
1.00%	pgbench	[kernel.vmlinux]	[k] _raw_spin_lock
0.87%	pgbench	libcrypto.so.1.0.2k	[.] sk_is_sorted
0.83%	pgbench	[kernel.vmlinux]	[k] try_to_wake_up
0.82%	pgbench	[kernel.vmlinux]	[k] futex_wake
0.80%	pgbench	[kernel.vmlinux]	[k] select_task_rq_fair
0.57%	pgbench	libpthread-2.17.so	[.] __pthread_mutex_lock
0.56%	pgbench	[kernel.vmlinux]	[k] available_idle_cpu

```
perf record -p $pid_of_pgbench --call-graph dwarf sleep 1
perf report --no-children
```

```
Samples: 19K of event 'cycles:ppp', Event count (approx.): 140614875678
Overhead Command Shared Object Symbol
- 68.39% pgbench [kernel.kallsyms] [k] queued_spin_lock_slowpath
- queued_spin_lock_slowpath
- 42.11% futex_q_lock
  futex_wait_setup
  futex_wait
  do_futex
  __x64_sys_futex
  do_syscall_64
- entry_SYSCALL_64_after_hwframe
- 41.59% __lll_lock_wait
  _L_lock_883
- pq_lockingcallback
- 17.27% CRYPTO_add_lock
  int_thread_release
- int_thread_get_item
- 16.91% ERR_get_state
- 11.34% ERR_clear_error
+ 7.33% pgtls_read
+ 4.01% pgtls_write
+ 5.57% get_error_values
```

openssl lock thread



 All

 Shopping

 Images

 Videos

 News

 More

Tools

About 485,000 results (0.32 seconds)

<https://www.openssl.org> › [blog](#) › [blog](#) › [2017/02/21](#) › t... 

OpenSSL and Threads

Feb 21, 2017 — In short, **OpenSSL** has always, and only, supported the concept of **locking** an object and sometimes it **locks** its internal objects. Read on for more ...

<https://www.openssl.org> › [docs](#) › [man1.0.2](#) › [man3](#) › th... 

threads - OpenSSL

OpenSSL can generally be used safely in multi-threaded applications provided that at least two callback functions are set, the `locking_function` and ...

<https://stackoverflow.com> › [questions](#) › [how-do-i-write-...](#) 

How do I write thread-safe OpenSSL code the new way?

Apr 26, 2020 · 1 answer

Actually, you no longer need to set up **locks** in **OpenSSL** 1.1.0 and later. Programming with **OpenSSL** Is **OpenSSL** thread-safe?

Tutorial on Using **OpenSSL** with pthreads [closed] - Stack ... Nov 19, 2011

Do I need to use CRYPTO **locking** functions for **thread** safety ... Oct 3, 2019

Using **OpenSSL** in a multi-threaded application - c++ - Stack ... Jun 28, 2012

Multithreaded program using **OpenSSL** and **locks** randomly ... Mar 17, 2017

[More results from stackoverflow.com](#)

OpenSSL Blog

[Blog](#) | [Archives](#)

POSTED BY RICH SALZ , FEB 21ST, 2017 11:00 AM

OpenSSL and Threads

This post talks about OpenSSL and threads. In particular, using OpenSSL in multi-threaded applications. It traces through the history, explains what was changed for the **1.1.0** release, and will hopefully provide some guidance to developers.

openssl 1.02 vs 1.1/3

- < 1.1: 112k tps
- 2800% CPU
- no ssl: 770k tps
- 1200% CPU
- >= 1.1: 560k tps
- 1300% CPU
- no tcp: 1.1M tps
- 910% CPU

```
perf record -p $some_backend_pid sleep 1  
perf report --sort comm,dso
```

Samples: 3K of event 'cycles', Event count (approx.): 2921279139

Overhead	Command	Shared Object
83.91%	postgres	postgres
6.47%	postgres	<u>[kernel.vmlinux]</u>
5.47%	postgres	libc-2.33.so
3.23%	postgres	<u>plpgsql.so</u>
0.33%	postgres	[vdso]
0.26%	postgres	<u>auto explain.so</u>
0.26%	postgres	<u>pg_stat_statements.so</u>
0.05%	postgres	<u>libm-2.33.so</u>
0.03%	postgres	libpthread-2.33.so

bpftrace

bpfttrace

- attach actions to
 - {kernel, userspace} tracepoints
 - {kernel, userspace} dynamic function probes
 - intervals, begin, end, sampling
- actions execute C-like code to build maps or print statements
- actions execute in kernel, in a [somewhat] sandboxed way

bpfttrace

- References:
 - bpfttrace manual:
https://github.com/iovisor/bpfttrace/blob/master/docs/reference_guide.md
 - list of postgres tracepoints:
<https://www.postgresql.org/docs/current/dynamic-trace.html>
 - Brendan Gregg blog post:
<https://www.brendangregg.com/ebpf.html#bpfttrace>
- Repos:
 - Tools in this talk: <https://github.com/anarazel/pg-bpfttrace>
 - flamegraph generation: <https://github.com/brendangregg/FlameGraph/>

Cache Hit Ratio

```
usdt:$1 postgresql:smgr__md__read__start  
{  
    @in_read[tid] = 1;  
}  
usdt:$1:postgresql:smgr__md__read__done / @in_read[tid] >= 1 /  
{  
    delete(@in_read[tid]);  
}  
tracepoint:iomap:iomap_readahead / @in_read[tid] /  
{  
    @cache_misses = count();  
    @cache_miss_bytes = sum(((uint64)args->nr_pages * 4096));  
}
```

Cache Hit Ratio

```
interval:s:1
{
    print(@cache_misses);
    print(@cache_miss_bytes);
    clear(@cache_misses);
    clear(@cache_miss_bytes);
}

END { clear(@in_read); }
```

<https://github.com/anarazel/pg-bpftrace/blob/main/pg-cache-hit.bt>

Cache Hit Ratio

...

@cache_misses: 0

@cache_misses: 48054

@cache_miss_bytes: 427405312

@cache_misses: 40262

@cache_miss_bytes: 377733120

@cache_misses: 31699

@cache_miss_bytes: 335884288

...

<https://github.com/anarazel/pg-bpftrace/blob/main/pg-cache-hit.bt>

Cache Hit Ratio by PID

```
tracepoint:iomap:iomap_readahead / @in_read[tid] /  
{  
    @cache_misses[pid] = count();  
    @cache_miss_bytes[pid] = sum(((uint64)args->nr_pages * 4096));  
}
```

<https://github.com/anarazel/pg-bpftrace/blob/main/pg-cache-hit-pid.bt>

bptrace

- information about a tracepoint:

```
bpfttrace -v -l 'tracepoint:block:block_rq_issue'
```

- list user space tracepoints:

```
bpfttrace -v -l 'usdt:/usr/lib/postgresql/14/bin/postgres:*'
```

Stacks

...

usdt:\$1:postgresql:lwlock__wait__done

{

 \$delta = nsecs - @lock_start[tid];

 delete(@lock_start[tid]);

 @[ustack()] = sum(\$delta);

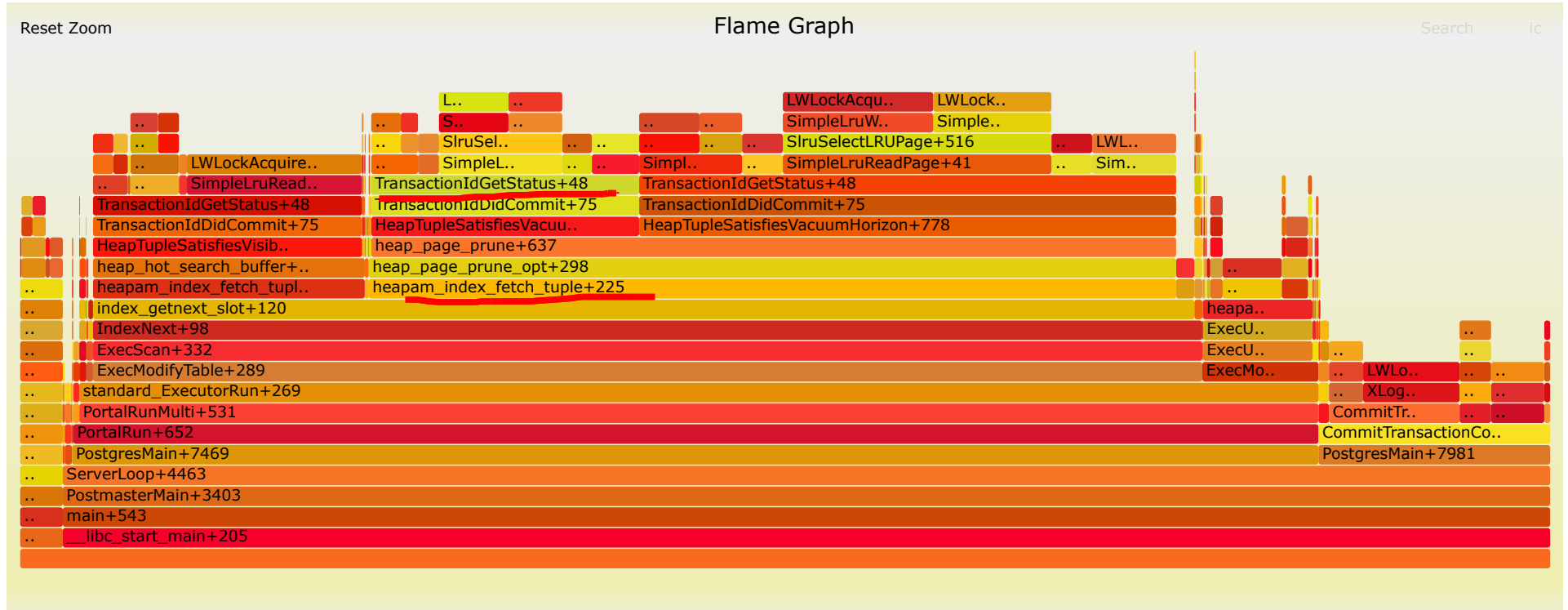
}

...

<https://github.com/anarazel/pg-bpftrace/blob/main/pg-lwlock-wait.bt>

<https://github.com/anarazel/pg-bpftrace/blob/main/pg-lwlock-stack.bt>

Stacks



pg_stat_statements

- tracks per-query statistics
 - “query fingerprints”
- <https://www.postgresql.org/docs/current/pgstatstatements.html>
- lacks many interesting statistics – too expensive, non-portable, etc
 - network input / output
 - lock wait time / congestion
 - cache hit ratio including kernel

pg_stat_statements + BPF!

```
uprobe:$2:pgss_store / arg1 != 0 /  
{  
    $queryid = arg1;  
  
    if (@pgss_send[tid, @pgss[tid]] > 0)  
    {  
        @stat_send[$queryid] = sum(@pgss_send[tid, @pgss[tid]));  
        delete(@pgss_send[tid, @pgss[tid]]);  
    }  
    ...
```

Network IO

```
tracepoint:syscalls:sys_exit_recvfrom
    / @pgss[tid] > 0 /
{
    if (args->ret > 0)
    {
        @pgss_recv[tid, @pgss[tid]] +=
            ((uint64) args->ret);
    }
}
```

```
tracepoint:syscalls:sys_exit_sendto
    / @pgss[tid] > 0 /
{
    if (args->ret > 0)
    {
        @pgss_send[tid, @pgss[tid]] +=
            ((uint64) args->ret);
    }
}
```

pg_stat_statements + BPF!

```
SELECT socket_send,  
       disk_read AS disk_bytes_read,  
       shared_blks_read * 8192 AS shared_bytes_read,  
       substr(query, 1, 25)  
FROM pg_stat_statements_extra  
WHERE dbid = (SELECT oid FROM pg_database WHERE datname = current_database())  
ORDER BY disk_read DESC NULLS LAST;
```

socket_send	disk_bytes_read	shared_bytes_read	substr
382664704	163119104	1141833728	COPY public.large (id, da
51159040	15073280	105512960	COPY public.pgbench_accou
35717120	8126464	56885248	COPY public.pgbench_histo
(null)	1171456	8200192	COPY public.pgbench_branc

https://github.com/anarazel/pg-bpfttrace/blob/main/pg_stat_statements.bt

Analyzing postgres performance problems using perf and eBPF

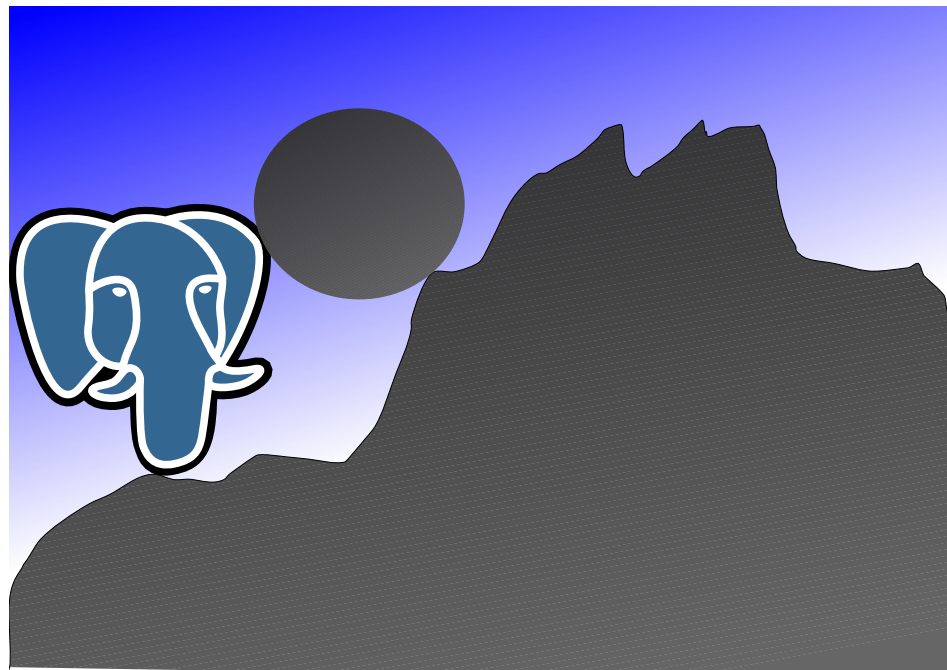
Andres Freund

PostgreSQL Developer,
Committer and Core Team
Member

andres@anarazel.de

andres.freund@microsoft.com

[@AndresFreundTec](#)



<https://anarazel.de/talks/2022-04-12-cituscon/perf-bpf.pdf>