OpenSCG

Debug and Tune PL/pgSQL Function

22 Feb 2018 | Bengaluru

Mohamed Tanveer
Senior PostgreSQL DBA, OpenSCG

pgConf
India 2018

about us

- Started Operations in early 2010
- Around 130 member team
- Headquartered in East Brunswick, NJ
- Presence in
  - Rochester, NY
  - San Mateo, CA
  - Hyderabad, India
  - Bangalore, India
- PostgreSQL Experts
- Long Community Involvement & Leadership
- AWS Consulting Partner

pl/pgsql

- it's simple...
  - SQL Procedural language
- It's functional...
  - fully integrated to PostgreSQL engine
  - server side computation
  - maintenance functions
- it's challenging...
  - debugging complex and nested functions

under the hood

- PL/pgSQL is like every other “loadable, procedural language.”
- When a PL function is executed, the fmgr loads the language handler and calls it.
- The language handler then interprets the contents of the pg_proc entry for the function (proargtypes, prorettype, prosrc).

how it works

- The PL/pgSQL statement tree is very similar to a PostgreSQL parse or execution tree.
- The call handler then executes that statement tree.
- On the first execution of a statement node, that has an SQL query in it, that query is prepared via SPI.
- The prepared plan is then executed for every invocation of that statement in the current session.

situation is !!!

psql

```
postgres=# explain analyze select * from my_func();
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Function Scan on getrecords  (cost=0.25..0.26 rows=1 width=4) (actual time=0.331..0.331 rows=1 loops=1)
 Total runtime: 0.365 ms
(2 rows)
```

OpenSCG

BIGSQL

agenda

- Options to log messages and errors
- How to capture the execution plan of a function
- How to profile a pl/pgsql function
- Limitations
- Questions ?

logging messages and error

.. using raise command

conf

```
# Level : DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, LOG, NOTICE, WARNING, ERROR, FATAL & PANIC
log_min_messages = ’WARNING’
# Level : DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL & PANIC
client_min_messages = ‘NOTICE’
```

pl/pgsql

```
RAISE NOTICE '%',to_char(clock_timestamp(), 'YYYY-MM-DD HH24:MI:SS.US');
```

.. using effect of a sql statement

```
SQL STATEMENT;
GET DIAGNOSTICS _row_count = ROW_COUNT;
RAISE NOTICE '%', _row_count ;
IF _row_count > 0 THEN
        ...
END IF;
```

pl/pgsql

The second method to determine the effects of a command is to check the special variable named FOUND, which is of type boolean. FOUND is a local variable within each PL/pgSQL function; any changes to it affect only the current function. It is set by SELECT INTO, PERFORM, UPDATE, INSERT, DELETE, FOR, FOREACH statements.

OpenSCG
BIGSQL

.. using call stack (9.3)

pl/pgsql

```
exception when others then
    GET STACKED DIAGNOSTICS
        _message = message_text,
        _detail = pg_exception_detail,
        _hint = pg_exception_hint;
    raise notice 'message: %, detail: %, \nhint: %', _message, _detail, _hint;
```

.. using call stack (9.4)

pl/pgsql

```
CREATE OR REPLACE FUNCTION inner_func() RETURNS integer AS  $$
DECLARE
        stack text;
BEGIN
        GET DIAGNOSTICS stack = PG_CONTEXT;
        RAISE NOTICE e'--- Call Stack ---\n%', stack;
        RETURN 1;
END;
$$ LANGUAGE plpgsql;
```

.. using exception block

pl/pgsql

```
CREATE OR REPLACE FUNCTION check_division() RETURNS INTEGER AS $$
DECLARE x INTEGER;
BEGIN
    x = 1 / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'division by zero';
    RETURN x;
END;
$$ LANGUAGE plpgsql;
```

capturing execution plan

.. using auto_explain

conf

```
shared_preload_libraries = 'auto_explain'
auto_explain.log_min_duration = '30s'
auto_explain.log_analyze = true
auto_explain.log_buffers = true
auto_explain.log_timing = true
auto_explain.log_verbose = true
auto_explain.log_nested_statements = true
auto_explain.log_format = 'text'
```

psql

```
postgres=# LOAD 'auto_explain';
postgres=# SET auto_explain.log_min_duration = '30s';
```

explain plan visualizer

https://explain.depesz.com/

http://tatiyants.com/pev/#/plans/

tracking execution statistics

.. using pg_stat_statements

conf

```
shared_preload_libraries = 'pg_stat_statements'

track_activity_query_size = 2048

pg_stat_statements.track = all

pg_stat_statements.max = 10000
```

SQL

```
SELECT (total_time / 1000 / 60) as total_minutes,
        (total_time/calls) as average_time,
        query
        FROM pg_stat_statements
ORDER BY 1 DESC LIMIT 10;
```

plan caching with pgBouncer

Prepared statements potentially have the performance advantage when a single session is being used to execute a large number of similar statements.

conf

```
# default

server_reset_query = 'DISCARD ALL'

# recommended

server_reset_query = "SET SESSION AUTHORIZATION DEFAULT;RESET ALL;DEALLOCATE ALL;CLOSE ALL;UNLISTEN *;SELECT pg_advisory_unlock_all();DISCARD SEQUENCES;DISCARD TEMP;"
```

plprofiler

plprofiler installation on linux

$ shell

```
$ sudo yum install python-setuptools
$ cd /usr/pgsql-9.6/share/contrib/
$ wget https://bitbucket.org/openscg/plprofiler/get/7ae3350da057.zip
$ unzip 7ae3350da057.zip
$ mv openscg-plprofiler-7ae3350da057/ plprofiler
$ cd plprofiler
$ export PATH=/usr/pgsql-9.6/bin:$PATH:$HOME/bin
$ make USE_PGXS=1
$ make USE_PGXS=1 install
$ cd python-plprofiler
$ python ./setup.py install
```

plprofiler installation on linux/macos using bigsql

$ shell

```
$ cd $HOME/bigsql
$ ./pgc update
$ ./pgc install plprofiler3-pg96
$ cd pg96
$ source pg96.env
$ plprofiler --help
```

plprofiler installation on windows using bigsql

cmd

```
cd C:\Program Files\PostgreSQL\
pgc update
pgc install plprofiler3-pg96
cd pg96
pg96-env.bat
set PATH=%PATH%;%PGC_HOME%\python2
cd bin
python plprofiler_util.py --help
```

profiling pl/pgsql function with plprofiler

running plprofiler

psql

```
postgres=# create extension plprofiler ;
CREATE EXTENSION
```

$ shell

```
plprofiler run --command "SELECT * from my_func();" --output out.html -U postgres -d postgres -h localhost
```

PL Profiler Report for current

PL/pgSQL Call Graph

PL Profiler Report for current

public.my_func() oid=3147017656

List of functions detailed below

- public.my_func() oid=3147017656

All 1 functions (by self_time)

Function public.my_func() oid=3147017656 (hide)

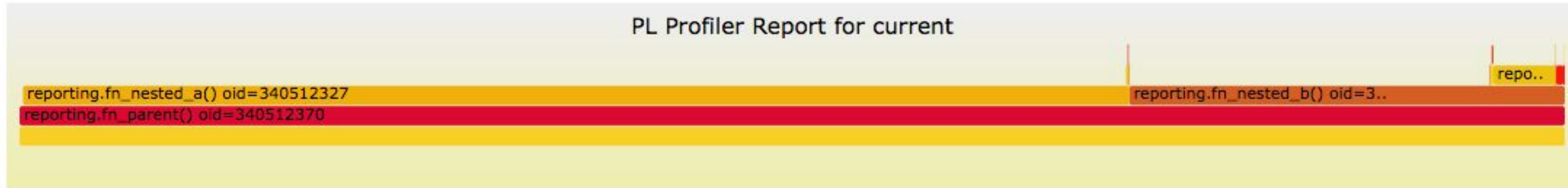
self_time = 1,086 μs
total_time = 1,086 μs

```
public.my_func ()
    RETURNS void
```

Line	exec_count	total_time	longest_time	Source Code
0	1	1,086 µs (100.00%)	1,086 µs	-- **Function Totals**
1	0	0 µs (0.00%)	0 µs	
2	0	0 µs (0.00%)	0 µs	BEGIN
3	1	365 µs (33.61%)	365 µs	CREATE TABLE IF NOT EXISTS foo (a INT);
4	1	708 µs (65.19%)	708 µs	FOR i IN 0..9 LOOP
5	10	578 µs (53.22%)	423 µs	INSERT INTO foo (a) VALUES (i);
6	0	0 µs (0.00%)	0 µs	END LOOP;
7	0	0 µs (0.00%)	0 µs	END
8	0	0 µs (0.00%)	0 µs	

use cases

profiling a complex and nested function



profiling a complex and nested function

List of functions detailed below

- public.fn_calculate_date_time_duration() oid=340512699
- public.fn_time_converter() oid=340512706
- public.fn_to_local_time() oid=340512705
- reporting.fn_nested_c() oid=340512333
- reporting.fn_nested_a() oid=340512327
- reporting.fn_nested_d() oid=340512418
- reporting.fn_nested_e() oid=340512332
- reporting.fn_nested_f() oid=340512370
- reporting.fn_nested_g() oid=340512502
- reporting.fn_nested_b() oid=340512487

Top 10 functions (by self_time)

Function reporting.fn_nested_a() oid=340512327 (show)

self_time = 49,010,507 μs
total_time = 49,161,055 μs

profiling a complex and nested function

2242	19,074	23,661 µs	(0.02%)	41 µs	
2243	19,074	22,154 µs	(0.02%)	1,077 µs	
2244	0	0 µs	(0.00%)	0 µs	
2245	19,074	16,416 µs	(0.01%)	38 µs	
2246	0	0 µs	(0.00%)	0 µs	
2247	19,074	36,678,835 µs	(27.79%)	1,077,620 µs	TRUNCATE TABLE
2248	0	0 µs	(0.00%)	0 µs	
2249	19,074	907,982 µs	(0.69%)	2,753 µs	INSERT INTO
2250	0	0 µs	(0.00%)	0 µs	(
2251	0	0 µs	(0.00%)	0 µs	
2252	0	0 µs	(0.00%)	0 µs	
2253	0	0 µs	(0.00%)	0 µs	
2254	0	0 µs	(0.00%)	0 µs	
2255	0	0 µs	(0.00%)	0 µs	

profiling a complex and nested function

2242	19,074	23,741 µs	(0.02%)	33 µs	
2243	19,074	21,300 µs	(0.02%)	34 µs	
2244	0	0 µs	(0.00%)	0 µs	
2245	19,074	16,885 µs	(0.02%)	28 µs	;
2246	0	0 µs	(0.00%)	0 µs	
2247	19,074	2,308,765 µs	(2.06%)	915 µs	DELETE FROM _ ; -- TRUNCATE TABLE _
2248	0	0 µs	(0.00%)	0 µs	
2249	19,074	732,531 µs	(0.65%)	1,703 µs	INSERT INTO _
2250	0	0 µs	(0.00%)	0 µs	(
2251	0	0 µs	(0.00%)	0 µs	
2252	0	0 µs	(0.00%)	0 µs	
2253	0	0 µs	(0.00%)	0 µs	
2254	0	0 µs	(0.00%)	0 µs	
2255	0	0 µs	(0.00%)	0 µs	
2256	0	0 µs	(0.00%)	0 µs	

profiling a live production system

conf

```
# requires server restart
shared_preload_libraries = 'plprofiler'
```

$shell

```
plprofiler reset
plprofiler monitor --pid <PID> --interval 10 --duration 300
plprofiler report --from-shared --name long_running_query  --output long_running_query.html
```

limitations

- Avoid nesting of complex functions.
- Additional logging could increase the log file size and disk usage.And add overhead to the server.
- Plprofiler could add overhead to the execution of the function. It is advised to perform such activity only during off peak hours or low traffic hours. Impact could be kept minimal by profiling only one function at a time.

Questions ?

/*Thank You!*/

Twitter	*@tanveer_munavar*
Github	*github.com/m-tanveer*
LinkedIn	*https://www.linkedin.com/in/tanveermunavar*

OpenSCG