Profiling PlpgSQL Tools

Pavel Stěhule

Pavel Stěhule

- PostgreSQL extensions
  - **Orafce**, **plpgsql_check**
- Pager **pspg**
- Some patches to Postgres
  - Variadic arguments, default parameters,
    mixed and named notation for passing parameters
  - Functions **format**, **xmltable**, ...
- Some patches to PLpgSQL
  - **RETURN QUERY**
  - **GET STACKED DIAGNOSTICS**

Old history

- 1998 PostgreSQL 6.4

  PLpgSQL by Jan Wieck

- 2007 PostgreSQL 8.3

  PLpgSQL Debug API by Korry Douglas

- 2009 PostgreSQL 8.4

  track_functions, auto_explain

history

- 2011 plpgsql_lint
- 2012 plProfiler separated from plDebugger
- 2013 plpgsql_check
- 2015 plProfiler revitalized by OpenSGC
- 2016 plProfiler + GraphUI
- 2018 plpgsql_check + profiler

DBG API

- Set of callback functions used by PLpgSQL executor
  - func_init
  - func_begin
  - func_end
  - stmt_begin
  - stmt_end
  - error_callback
  - assign_expression

Motivation

- Profiling is important
- Knowleage
- Performance
- User experience
- Stability
- It is important to not be blind
- Good decisions needs good data

Source of issues

- Code complexity
- Missing knowleage about technology
- Bugs on app or sys side

What is important?

- Detect slow or lot of called routines
- Detect slow lines inside routines
- Detect slow queries used by routines

Example

```
CREATE OR REPLACE FUNCTION slow_one(n int, v int)
RETURNS int[] AS $$
DECLARE r int[] = ´{}´;
BEGIN
  FOR i IN 1..n
  LOOP
    r := r || v;
  END LOOP;
  RETURN r;
END;
$$ LANGUAGE plpgsql;
```

Why code can be slow?

- Lot of reasons
  - Update immutable structures
  - Unwanted casts (hidden)
  - Missing or blocked query optimizations
- **PLpgSQL is fast enough!**
  - **Usually (but it is not C)**

What is PLpgSQL?

- It is simple AST interpret
- AST interprets are very fast, but some instructions are not possible (Goto)
- Every expression is SELECT (currently not supported by JIT)
- It is designed be glue for SQL
- It is not designed for heavy calculations

What do want search?

- Often called routines
- Slow routines
- Slow queries

Example

```
CREATE TABLE users(id bigint PRIMARY KEY, …);

CREATE OR REPLACE FUNCTION slow_01(user_id numeric)
RETURNS void AS $$
BEGIN
  IF EXISTS(SELECT * FROM users WHERE id = user_id) THEN
    ...
```

Tools

- track_functions
- auto_explain (nested)
- plProfiler
- plpgsql_check

track_functions

```
SET track_function to ´all´;
…
postgres=# select * from pg_stat_user_functions ;
```

funcid	schemaname	funcname	calls	total_time	self_time
16699	public	fast_func	100	0.407	0.407
16701	public	test_func	1	1034.994	4.203
16698	public	slow_func	100	1030.383	1030.383

(3 rows)

auto_explain

- log_nested_statements to on
-

plpgsql_check (profiler)

```
load 'plpgsql_check';
set plpgsql_check.profiler to on;

select * from plpgsql_profiler_function_tb('slow_func');
```

plpgsql_check

- C extension
- Check validity of embedded SQL
- Check some performance issues
  (hidden casting, wrong flags)
- Check some security issues (SQL injection)
- Integrated profiler
- Designed for simple usage

plProfiler

```
plprofiler run --command "select test_func(100)" \
   --output test_func.html -h localhost
```

plProfiler

- C extension, Python code for reports
- Simple usage – results has html formats
- Just profiler
- Build call graphs

Example

```
CREATE OR REPLACE FUNCTION district_name(dc varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT name FROM districts WHERE code = dc);
END;

SELECT town FROM towns WHERE district_name(district) = ´Krasnoarmejsk´;
```

Example

```
CREATE OR REPLACE FUNCTION district_name(dc varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT name FROM districts WHERE code = dc);
END;

SELECT town FROM towns WHERE district_name(district) = ´Krasnoarmejsk´;
```

BAD

WHY?

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END;

SELECT town FROM towns WHERE code = district_code(´Krasnoarmejsk´);
```

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END;

SELECT town FROM towns WHERE code = district_code(´Krasnoarmejsk´);
```

BAD

WHY?

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END STABLE;

SELECT town FROM towns WHERE code = district_code('Krasnoarmejsk');
```

Why some queries are fast on Oracle and not on PG?

- Different planners (estimators)

  some patterns are better supported by Oracle (nvl, aggpush down, ..), some ugly patterns are supported by O.

- Different optimizers

  from_collapse_limit, join_collapse_limit, GEQO

- Missing implicit plan cache on Postgres

- Slower start of queries on Postgres

  due simple and more dynamic implementation

- Extra intensive UPDATEs are slower on Postgres