Injection Points

PostgreSQL Conference Japan 2024
December 6th, Tokyo
Michael Paquier

The lecturer

- Michael Paquier.
- French, based in Tokyo.
- PostgreSQL contributor since 2009
  - Committer
  - Hacker, Blogger, CF manager
- Twitter: @michaelpq
- Website: https://paquier.xyz

Agenda

- Concept
- Core facilities
- Examples

Concept

Postgres makes testing hard

- Multi-process infrastructure.
- Cannot really synchronize actions.
- Some tricks
  - Locking (LOCK pg_database?)
  - Hooks, limited by code paths and states.
  - Extensions, limited in backend to external libraries.

Requirements

- Developer tool
  - Run custom code
  - At custom location
- Properties
  - Flexible: core, extension, back-patch.
  - Customizable, extensible
  - Cheap
  - Maintenance

Test Designs

- Race Conditions
- Force states:
  - ERROR, FATAL, PANIC
  - Dynamic Stack Manipulation
- Cross-process interactions:
  - Wait and Wake
  - Monitoring
- Print information

Past proposals

- Better infra for concurrency (2020)
  https://postgr.es/m/CAPpHfdtSEOHX8dSk9Qp+Z++i4BGQoffKip6JDWngEA+g7Z-XmQ@mail.gmail.com
- Forks of upstream:
  - PROBE_POINT() by 2ndQ:
    https://postgr.es/m/CAGRY4nyiBqP=nNmYbee1q2rcyWROM68N36DiEF4ttv3Zr9Wocw@mail.gmail.com
  - Greenplum (CloudberryDB), proposed once upstream.
  - Probably most products do that anyway.

Code

- Documentation: https://www.postgresql.org/docs/devel/xfunc-c.html#XFUNC-ADDIN-INJECTION-POINTS
- Disabled by default
  - configure --enable-injection-points
  - meson -Dinjection_points=true
- Sources
  - src/backend/utils/misc/injection_point.c
  - src/include/utils/injection_point.h
- Shared memory array (point, library, function)
- Process-level cache with hash table: loaded function pointer.
- Can be used in postmaster process
- Enabled in CI, CFBot (http://cfbot.cputube.org/)

Basics

- INJECTION_POINT(“hoge”)

```
#ifdef USE_INJECTION_POINTS
#define INJECTION_POINT(name) InjectionPointRun(name)
#else
#define INJECTION_POINT(name) ((void) name)
#endif
```

```
extern void InjectionPointAttach(const char *name,
                                 const char *library,
                                 const char *function,
                                 const void *private_data,
                                 int private_data_size);
extern bool InjectionPointDetach(const char *name);
extern void InjectionPointRun(const char *name);
```

Critical sections

- Problem with internal_load_library()@dfmgr.c
- Allocation when running a point the first time!
- Crashes:

```
START_CRIT_SECTION();
INJECTION_POINT(“hoge”);    /* PANIC */
END_CRIT_SECTION();
```

Cached points

- Two-step process:
  - Load a point, heat process cache.
  - Run callback from cache.
- Works:

```
INJECTION_POINT_LOAD(“hoge”);
START_CRIT_SECTION();
INJECTION_POINT_CACHED(“hoge”);    /* okay */
END_CRIT_SECTION();
```

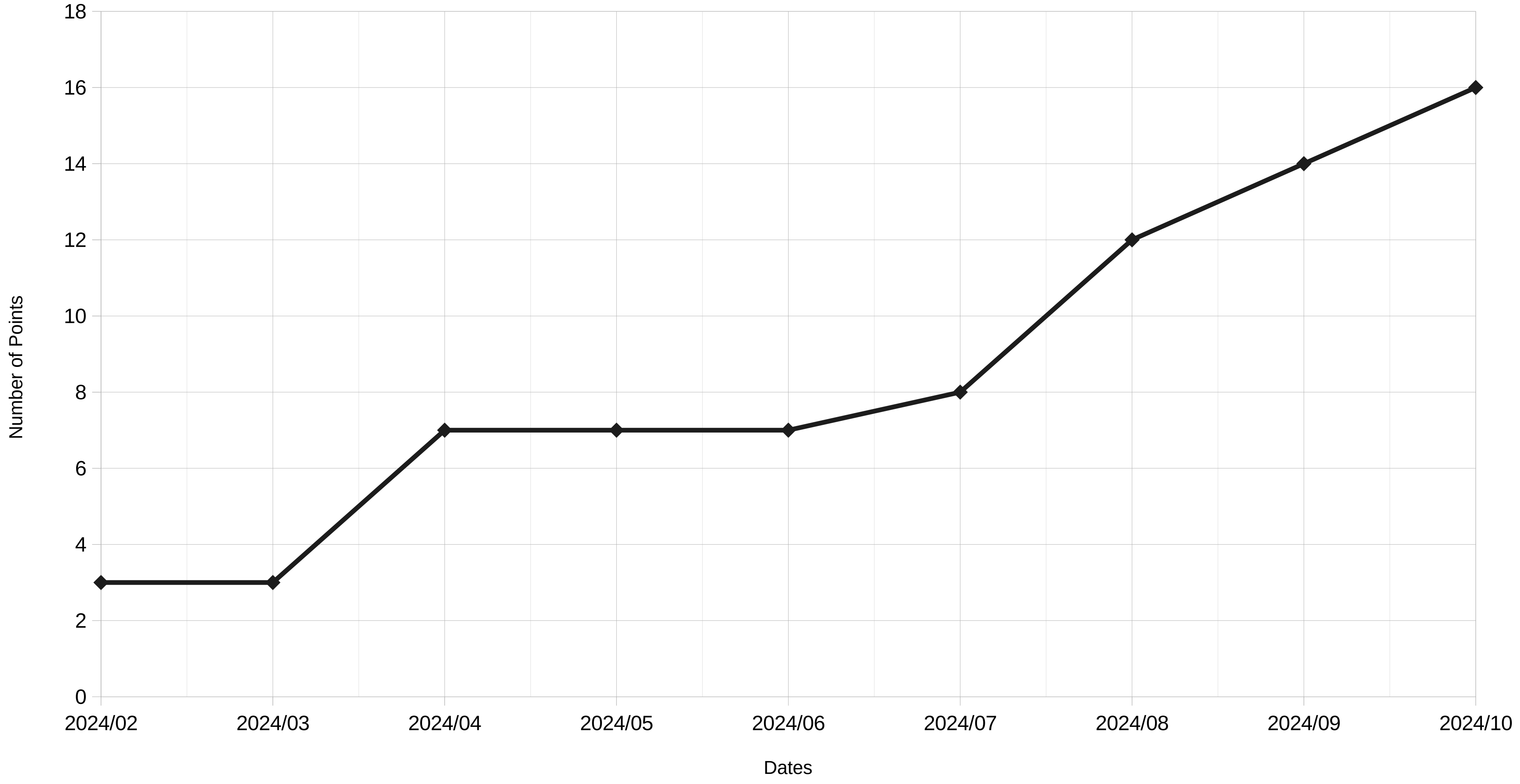
Stack Manipulation

- No runtime arguments given to callback!
- Conditional states:

```
If (IS_INJECTION_POINT_ATTACHED(“hoge”))
{
    /* Force a custom state */
    some_var = 199;

    /* Execute callback */
    INJECTION_POINT_CACHED(“hoge”);
}
```

INJECTION_POINT() vs Time



Core Facilities

injection_points

- src/test/modules/injection_points/
- Not contrib/
  - May evolve in stable branches
  - Flexible for bug fixes (perhaps critical)
- SQL functions:
  - Attach, Detach
  - Wakeup
  - Run, Load
  - Callbacks: ‘error’, ‘notice’, ‘wait’.

injection_points - Objects

```
=# CREATE EXTENSION injection_points;
=# \dx+ injection_points
    Objects in extension "injection_points"

               Object description

------------------------------------------------
 function injection_points_attach(text,text)
 function injection_points_cached(text)
 function injection_points_detach(text)
 function injection_points_load(text)
 function injection_points_run(text)
 function injection_points_set_local()
 function injection_points_wakeup(text)
(7 rows)
```

injection_points – Local points

- SQL tests, mostly.
- injection_points_set_local() at top of SQL script
- Concurrent-safe
- Automatic detach at process shutdown.

```
SELECT injection_points_set_local();
SELECT injection_point_attach(‘foo’, ‘error’);
SELECT injection_point_run(‘foo’); -- ERROR!
-- Or insert here SQL sequence able to run ‘foo’
\q -- detach
```

Isolation Tests

- isolationtester, relies on wait&wakeup.

```
setup    { CREATE EXTENSION injection_points; }
teardown { DROP EXTENSION injection_points; }

session s1
setup   {
  SELECT injection_points_set_local();
  SELECT injection_points_attach('injection-points-wait', 'wait');
}
step wait1 { SELECT injection_points_run('injection-points-wait'); }

session s2
step wakeup2    { SELECT injection_points_wakeup('injection-points-wait'); }
step detach2    { SELECT injection_points_detach('injection-points-wait'); }

# Detach after wait and wakeup.
permutation wait1 wakeup2 detach2
```

TAP tests

- Wait and Wake, mostly!
- Monitoring: pg_stat_activity, server logs.

```
$node->start;
[...]
# Wait for specific event.
$node->wait_for_event($backend_type, $event_name);
[...]
# Hold session, then wait for specific event.
my $observer = $node->background_psql('postgres');
$observer->query_until(
  qr/start/,
  q{\echo start
    SELECT injection_points_run($event_name);});
$node->wait_for_event('client backend', $event_name);
```

Examples

SQL - REINDEX bug

- Index corruption: “safe”? Not really.
  - Expressions and predicates.
  - Possible access to other tables.
  - Snapshot waits missed.
- reindex_conc.sql in injection_points module.
- Commit 5bbdfa8a18dc.

SQL - REINDEX bug

```
--- a/src/backend/commands/indexcmds.c
+++ b/src/backend/commands/indexcmds.c
[...]
@@ -3782,8 +3783,16 @@ ReindexRelationConcurrently(...)
+#ifdef USE_INJECTION_POINTS
+       if (idx->safe)
+           INJECTION_POINT("reindex-conc-index-safe");
+       else
+           INJECTION_POINT("reindex-conc-index-not-safe");
+#endif
```

```
-- Attach a NOTICE to both points
SELECT injection_points_attach('reindex-conc-index-safe', 'notice');
SELECT injection_points_attach('reindex-conc-index-not-safe', 'notice');
-- Check the state with the commands
REINDEX INDEX CONCURRENTLY reindex_inj.ind_simple;
NOTICE:  notice triggered for injection point reindex-conc-index-safe
REINDEX INDEX CONCURRENTLY reindex_inj.ind_expr;
NOTICE:  notice triggered for injection point reindex-conc-index-not-safe
```

Isolation - GRANT/VACUUM

- Data loss and index corruption.
  - In-place UPDATE for stats at the end of VACUUM.
  - GRANT TABLE, DATABASE
- Test “inplace” in injection_points module.
- Commit a07e03fd8fa7.

Isolation - GRANT/VACUUM

```
--- a/src/backend/access/index/genam.c
+++ b/src/backend/access/index/genam.c
@@ -6089,6 +6194,19 @@ heap_inplace_update(...)
[...]
+        INJECTION_POINT("inplace-before-pin");
```

```
session s1
setup   {
  SELECT injection_points_set_local();
  SELECT injection_points_attach('inplace-before-pin', 'wait');
}
step vac1       { VACUUM vactest.orig50;  -- wait during inplace update }

# Transactional updates of the tuple vac1 is waiting to inplace-update.
session s2
step grant2     { GRANT SELECT ON TABLE vactest.orig50 TO PUBLIC; }
step revoke2    { REVOKE SELECT ON TABLE vactest.orig50 FROM PUBLIC; }
```

TAP - Promote and checkpoint

- Race condition:
  - restartpoint still running after promotion.
  - PANIC after restart.
- Wait for restartpoint in startup process, wakeup.
- Recovery test 041_checkpoint_at_promote.
- Bugfix at 7863ee4def65.
- Test at 6782709df81f, 2~3min vs 5s.

TAP - Promote and checkpoint

```
--- a/src/backend/access/transam/xlog.c
+++ b/src/backend/access/transam/xlog.c
@@ -7528,6 +7529,12 @@ CreateRestartPoint(int flags)
[...]
+    INJECTION_POINT("create-restart-point");
```

```
$node_standby->safe_psql('postgres',
   "SELECT injection_points_attach('create-restart-point', 'wait');");
[...]
my $psql_session =
   $node_standby->background_psql('postgres', on_error_stop => 0);
$psql_session->query_until(
   qr/starting_checkpoint/, q(\echo starting_checkpoint
   CHECKPOINT;));
[...]
$node_standby->wait_for_event('checkpointer', 'create-restart-point');
[...]
$node_standby->promote;
[...]
$node_standby->safe_psql('postgres',
   "SELECT injection_points_wakeup('create-restart-point');");
```

Thanks!
Questions?