第九届PostgreSQL中国技术大会

2019 PostgreSQL Conference China

开源驱动 自主研发

主办：PostgreSQL中文社区

协办：ITPUB

2019年11月29日-30日

北京维景国际大酒店

Fault injection to test PostgreSQL code

Asim R P
apraveen@pivotal.io
Pune, India

Agenda

- What is lacking in current testing frameworks?
- What is fault injection?
- Proposal to inject faults
- Tests to demonstrate proposed faultinjector patch

Testing frameworks in PostgreSQL

- pg_regress (src/test/regress)
  - A test is written as a .sql file
  - Expected output is saved as an answer file (.out)
  - pg_regress runs a .sql file, compares the output with the answer
- isolation (src/test/isolation)
  - tests concurrent transactions
  - .spec file - SQL commands within each transaction and how to interleave them

isolation: test that select waits for alter

```
session “s1”
setup { BEGIN ISOLATION LEVEL READ COMMITTED; }
step “alter1” { ALTER TABLE test ADD RENAME TO test2; }
step “commit1” { COMMIT; }
session “s2”
setup { SET default_transaction_isolation = read committed; }
step “select2” { SELECT * from test; }
permutation “alter1” “s2” “commit1”
```

Testing frameworks in PostgreSQL

- TAP - tests written in perl (Test::More)
  - Orchestrate a cluster - master and one or more standby
  - initdb, start / stop cluster, etc.
  - src/test/perl/README
- src/test/modules (not run in CI)
  - modular testing of a specific component, e.g. planner
  - src/test/modules/README

What is lacking in PostgreSQL tests?

- Test that a backend is killed after writing commit record but before updating the transaction status in pg_clog
- Is synchronous replication really synchronous?
- Was a cached plan reused?
- Anything comes to your mind?

Fault injection

- Unconference discussion at PGcon 2019 in Ottawa
- Already being used in Greenplum
- Patch proposed on pgsql-hackers:
  - https://www.postgresql.org/message-id/flat/CANXE4TdxdESX1jKw48xet-5GvBFVSq%3D4cgNeioTQff372KO45A%40mail.gmail.com

Fault point (./configure CPPFLAGS=-DFAULT_INJECTOR)

```
--- a/src/backend/executor/execIndexing.c
+++ b/src/backend/executor/execIndexing.c
@@ -289,6 +289,7 @@ ExecInsertIndexTuples(TupleTableSlot *slot,
        bool            isnull[INDEX_MAX_KEYS];

        Assert(ItemPointerIsValid(tupleid));
+       SIMPLE_FAULT_INJECTOR("insert_index_tuples");

        /*
         * Get information from the result relation info structure.
```

Fault point - macro definition

```
#ifdef FAULT_INJECTOR
#define SIMPLE_FAULT_INJECTOR(FaultName) \
    FaultInjector_TriggerFaultIfSet(FaultName, "", "")
#else
#define SIMPLE_FAULT_INJECTOR(FaultName)
#endif
```

```
--- a/src/backend/access/heap/heapam.c
+++ b/src/backend/access/heap/heapam.c
@@ -1875,6 +1875,12 @@ heap_insert(Relation relation, HeapTuple tup, CommandId cid,
        Buffer          vmbuffer = InvalidBuffer;
        bool            all_visible_cleared = false;

+#ifdef FAULT_INJECTOR
+       FaultInjector_TriggerFaultIfSet(
+               "heap_insert",
+               "" /* database name */,
+               RelationGetRelationName(relation));
+#endif
        /*
         * Fill in tuple header fields and toast the tuple if necessary.
         *
```

Set a fault using its name

- CREATE EXTENSION faultinjector;
- Enable the “heap_insert” fault when a tuple is inserted into “mytable”
  - SELECT inject_fault(‘heap_insert’, ‘error’, ‘’, ’mytable’, …);
- Wait until the fault is triggered (blocking call)
  - SELECT wait_until_triggered_fault(‘heap_insert’, 1);
- Check the status
  - SELECT inject_fault(‘heap_insert’, ‘status’);
- Reset the fault
  - SELECT inject_fault(‘heap_insert’, ‘reset’);

More on inject_fault

inject_fault(… ,

<start_occurrence>,

trigger only after the fault point is reached as many number of times

<end_occurrence>,

stop after triggerring as many number of times (fault state completed)

<sleep_seconds>

seconds to sleep in a sleep fault)

inject_fault_remote()

- Same as inject_fault()
- Additional args: hostname and port number
- New libpq message to inject faults
- Useful for testing standby servers in a cluster

Fault types

- error: elog(ERORR) leads to transaction abort
- skip: do nothing - indicates that the fault point was reached, also used for custom action
- suspend
- reset
- status

```
--- a/src/backend/access/transam/xlog.c
+++ b/src/backend/access/transam/xlog.c
@@ -8537,6 +8537,14 @@ CreateCheckPoint(int flags)
        VirtualTransactionId *vxids;
        int                     nvxids;

+#ifdef FAULT_INJECTOR
+       if (SIMPLE_FAULT_INJECTOR("checkpoint") == FaultInjector_FaultTypeSkip)
+       {
+               /* Custom logic here ... */
+               return;
+       }
+#endif
+
        /*
         * An end-of-recovery checkpoint is really a shutdown checkpoint, just
         * issued at a different time.
```

Fault states

- injected but not triggered - SELECT inject_fault();
- completed - triggered maximum number of times, will no longer trigger
- triggered - reached during execution at least once

Faults offer fine grained control

- Enable complex testing scenarios
- Was a specific flag in shared memory set, and when?
- Was a branch in a function taken?
- Can be used in regress, isolation as well as TAP tests

Demonstrative test: speculative insert

```
CREATE TABLE test(key TEXT, data TEXT);

CREATE UNIQUE INDEX ON test(key);

T1: INSERT INTO test values ('k1', 'inserted T1') ON CONFLICT
DO UPDATE SET data = test.data || ' conflict update T1';

T2: INSERT INTO test values ('k1', 'inserted T2') ON CONFLICT
DO UPDATE SET data = test.data || ' conflict update T2';
```

Demonstrative test: speculative insert

Conflicts detected after a tuple is inserted into heap but before inserting into index are handled properly

- T1 goes first and inserts the tuple into heap
- Before T1 updates the index, T2 inserts into both, heap and the index.
- T2 sees no conflicts because T1 is still in progress.
- T1 moves ahead. T1 should detect a conflict and abort.

Demonstrative test: speculative insert

- Find it in the faultinjector patch:
  - src/test/isolation/specs/insert-conflict-with-faults.spec

Demonstrative test: synchronous replication

Ensure that commits on master block until a synchronous standby acknowledges the commit

- Master writes a commit record
- The backend process on master starts waiting for standby to flush WAL upto the commit LSN
- Standby goes down before the WAL receiver writes WAL upto the commit LSN

Demonstrative test: synchronous replication

- Find it in the faultinjector patch, it's a TAP test:
  - src/test/recovery/t/007_sync_rep.pl

THANKS