Fault injection facility for PostgreSQL hackers

Asim R P (Greenplum, VMware)
pasim@vmware.com

Overview

- How is PostgreSQL tested today?
- Interesting test scenarios
- Fault injection proposal
- Let’s write some tests using faults

src/test/regress

- Test case is a set of SQL statements (.sql file)
- Expected output (.out file)
- Execute a .sql file against PostgreSQL server under test
- Compare the results with expected output

src/test/isolation

- Interleaving concurrent sessions
- Steps within a session are defined as SQL statements
- A test case is permutations of the steps from multiple transactions

isolation spec

```
setup  { CREATE TABLE foo (…); }

teardown { DROP TABLE foo; }

session “s1”

setup { BEGIN; }

step “alter1” { ALTER TABLE foo …; }

step “commit1” { COMMIT; }

session “s2”

step “select2” { SELECT * FROM foo; }

permutation “alter1” “select2” “commit1”
```

TAP:
`find src -name t -type d`

- Perl power! (Test::More)
- Cluster orchestration - initdb
- Streaming replication
- Backup / restore
- Kill a backend with SIGQUIT, causing crash recovery

Interesting test scenarios

- Bring down a synchronous standby while a master backend waits for commit LSN to be flushed

- Server crash after writing commit WAL record but before updating commit log (clog)

- Streaming replication behaviour when the replication connection breaks intermittently — induce replay lag and break the connection

Interesting test scenarios

- Group update transaction status in CLOG during commit / abort
- Was syscache / relcache utilised or there was a cache miss?
- Any scenario you can think of?

Fault injection

- Fault point - point of interest in source code
- Defined by instrumenting the source code (`CPPFLAGS=-DFAULT_INJECTOR`)
- SQL interface to enable a pre-defined fault point
- Enabled fault points are remembered in shared memory
- Patch - Fault injection framework

Definition of “heap_insert” fault

```
--- 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.
         *
```

Enable fault using SQL

```
CREATE EXTENSION faultinjector;

SELECT inject_fault(

    ‘heap_insert’, ’error’,

    ‘’,’my_table’ …);
```

Fault actions

- error: elog(ERROR) leading to transaction abort
- skip: do nothing - used for custom action
- suspend / resume
- reset
- status - how many times triggered

Fault type “skip”

```
--- 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.
```

SQL interface

```
SELECT wait_until_triggered_fault(
   'heap_insert', 1);
SELECT inject_fault(
   'heap_insert', 'status');
SELECT inject_fault(
    'heap_insert', 'reset');
```

SQL interface

```
SELECT inject_fault_remote(
    'heap_insert', 'error',
    'standby_host', 5433);
```

SQL command is run on master, fault is injected on standby

Fault status

- Triggered - reached during execution and the right action was taken
- Injected but not triggered - inject_fault();
- Completed - triggered max number of times

Let’s write some tests

Speculative insert test

```
INSERT INTO … ON CONFLICT …;
```

- Conflicts detected after a tuple is inserted into heap but before it is inserted into index are handled correctly.
- Test without faults: src/test/isolation/specs/insert-conflict-specconflict.spec
- The test went through several iterations: pgsql-hackers discussion

Speculative insert test

```
setup {
    CREATE TABLE upserttest(key text, data text);
    CREATE UNIQUE INDEX ON upserttest(key);
    CREATE EXTENSION faultinjector;
    —— Suspend before inserting into index
    SELECT inject_fault(‘insert_index_tuples’, ‘suspend’);
    —— Ensure that a speculatively inserted tuple was killed
    SELECT inject_fault(‘heap_insert_speculative’, ‘skip’);
}
```

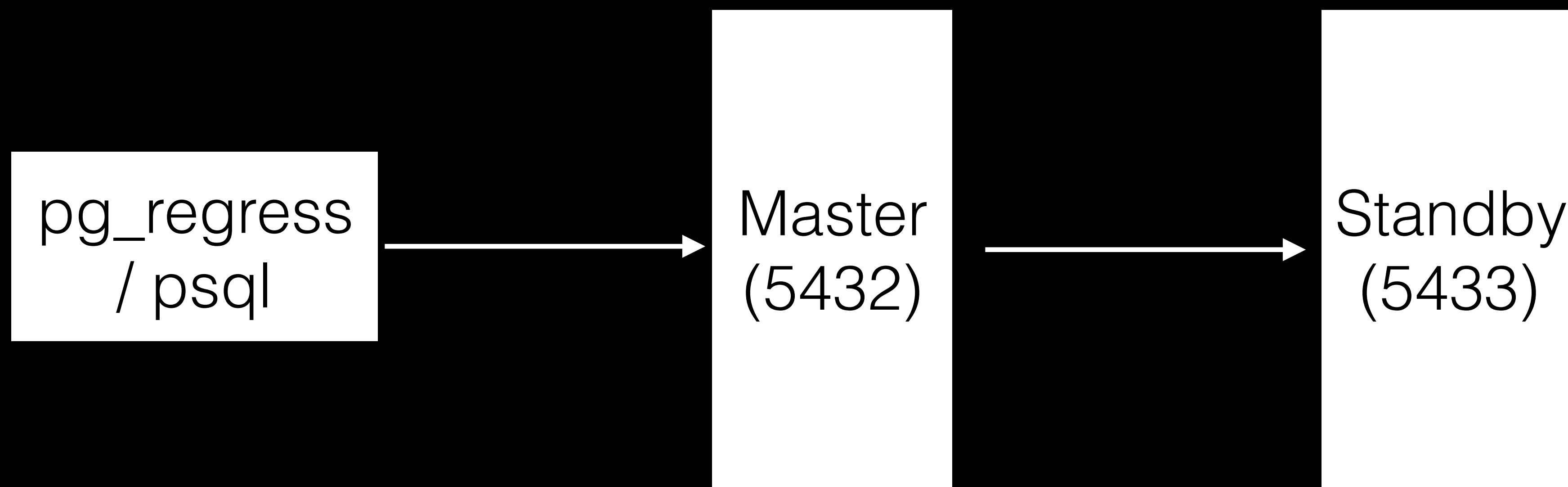
```
step "s1_upsert" {
    INSERT INTO upserttest(key, data)
    VALUES('k1', 'inserted s1') ON CONFLICT (key) DO UPDATE
    SET data = upserttest.data || ' updated by s1'; }
step "s2_upsert" {
    INSERT INTO upserttest(key, data)
    VALUES('k1', 'inserted s2') ON CONFLICT (key) DO UPDATE
    SET data = upserttest.data || ' updated by s2'; }
step "unblock_s1" {
    SELECT * FROM inject_fault(
        'insert_index_tuples', 'resume'); }
```

```
permutation
    # S1 should hit the suspend fault and
    # block before inserting new tuple into index
    "s1_upsert"&
    # S2 should insert without conflict
    "s2_upsert"
    “unblock_s1"
    # Validate that the skip fault
    # heap_abort_speculative was hit
    "s1_fault_status"
    "s1_select"
```

Streaming replication test

- Start with synchronous_commit = remote_write
- Build up replay lag and disconnect streaming replication connection
- Synchronous commit behaviour should not change upon reconnection
- It does change when standby lags in replay: discussion on pgsql-hackers

Setup




```
synchronous_commit = on

synchronous_standby_names = '*'
```

Streaming replication test

- Induce replay lag: inject fault on standby

  `recovery_min_apply_delay` is not helpful

- Terminate replication connection (simulate network blip): inject fault on standby

- Streaming replication should resume as soon as the connection is reestablished

It’s a pg_regress test

```
-- Induce 10 seconds delay per WAL record replay

select inject_fault('redo_main_loop', ‘sleep', …, 10,
'localhost', 5433);

insert into replay_lag_test values (‘before disconnect’);

-- Kill WAL receiver, it resumes itself

select inject_fault('wal_receiver_loop', 'fatal',
'localhost', 5433);

-- The insert should wait until standby confirms flush up
to commit LSN.  Wait should be <10 sec.

insert into replay_lag_test values ('after disconnect',
now());
```

Please review the patch!