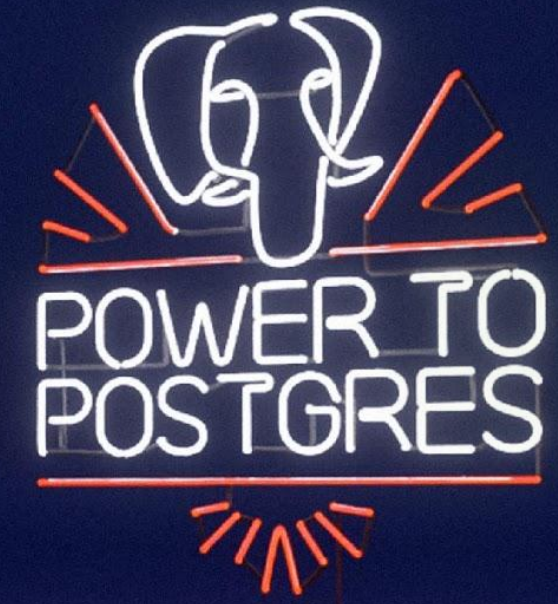
Global Deadlock Detection in Distributed PostgreSQL Environment

Koichi Suzuki

Mar. 11, 2021



EDB

Agenda

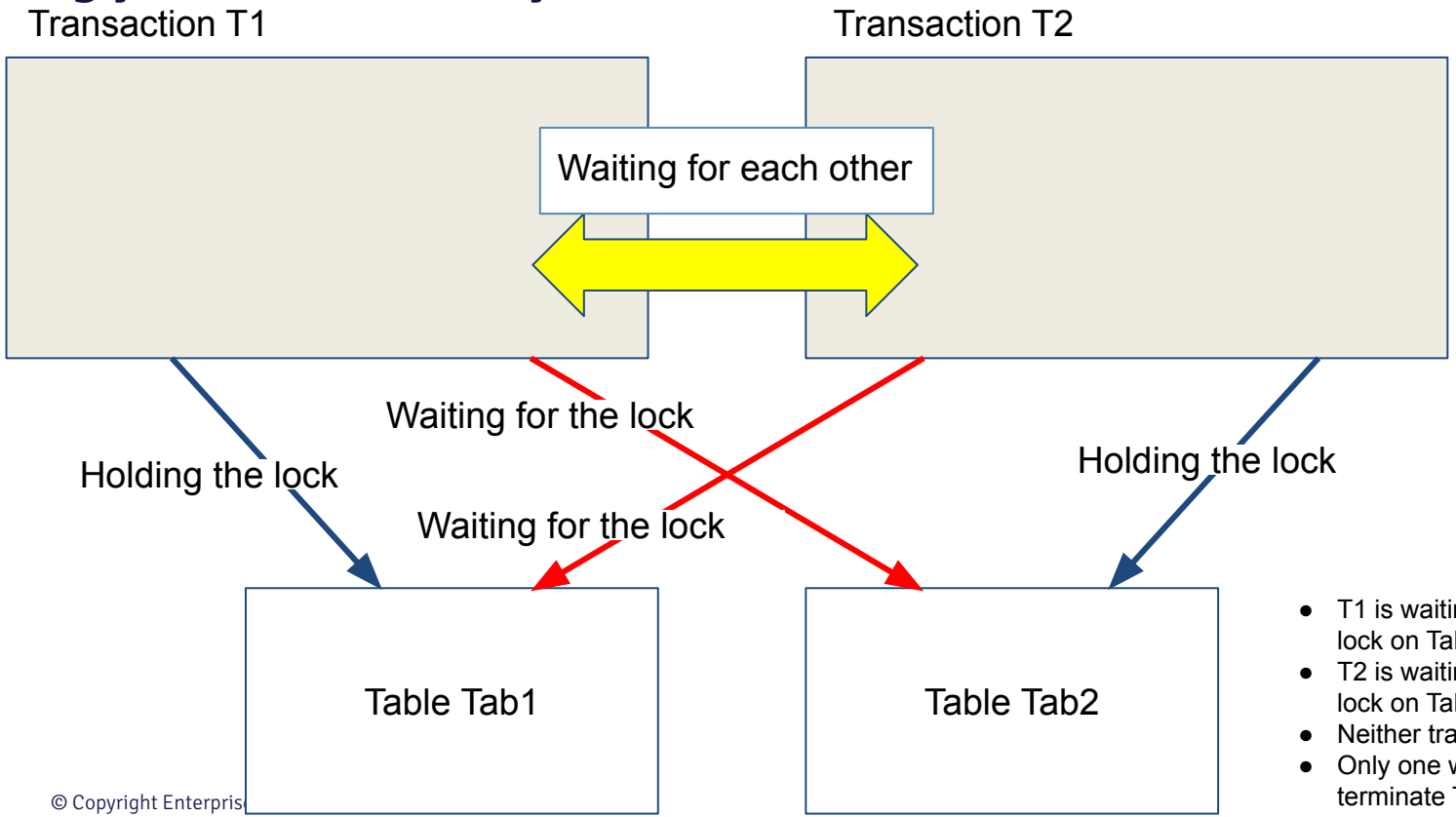
- Deadlock Overview
- Deadlock detection in local database
- Deadlock detection in database cluster
- Implementation architecture and actual

EDB

What is Deadlock?

- Database transaction obtains locks on rows/tables/other database objects to protect them from undesirable change by different transaction to maintain data consistency.
- When such rows/tables have already been locked, transaction must wait until such locks are released.
- Sometimes lock requests clinch between transactions. This situation is called “deadlock”.
- When a deadlock occurs, there is no way to proceed other than terminating one of the transactions involved.

Typical and simple deadlock situation



- T1 is waiting for T2 to release the lock on Table Tab2.
- T2 is waiting for T1 to release the lock on Table Tab1.
- Neither transaction can proceed.
- Only one way forward is to terminate T1 or T2 from outside.

How to detect the deadlock?

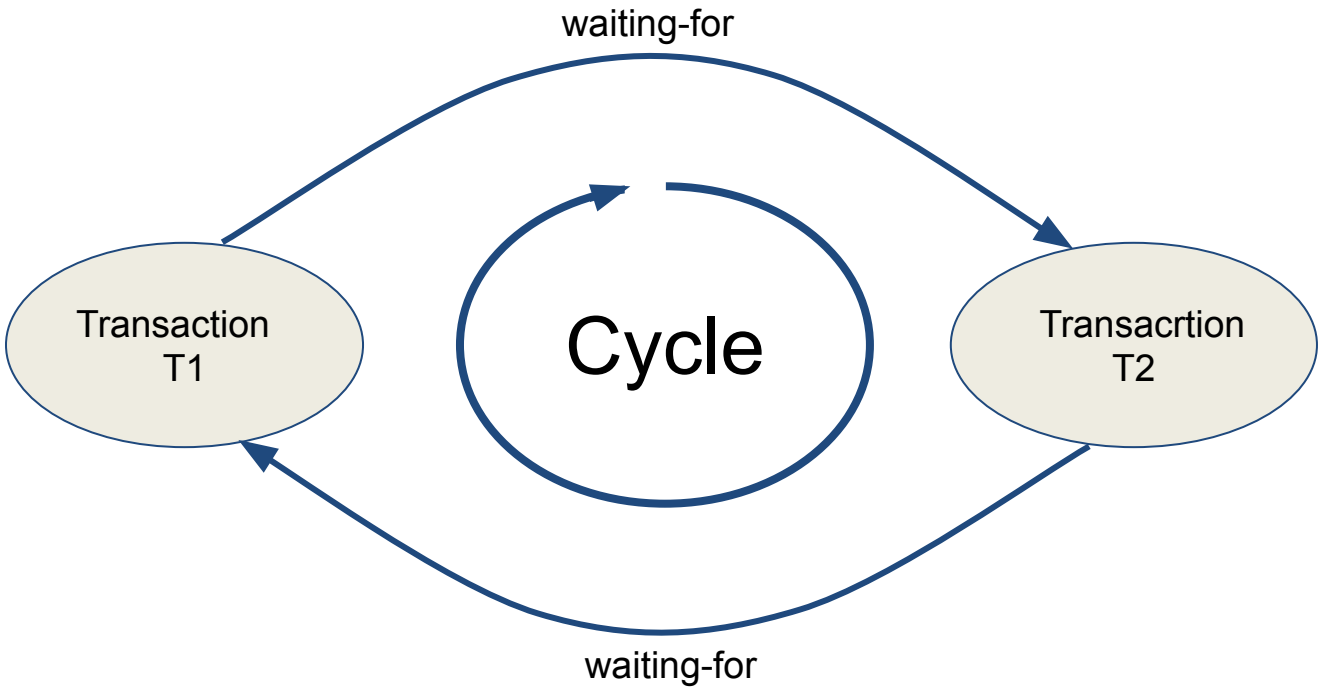
Wait-for-Graph is used

- Directed graph to represent what transaction is waiting for what

If there is a deadlock, there is a cycle in wait-for-graph

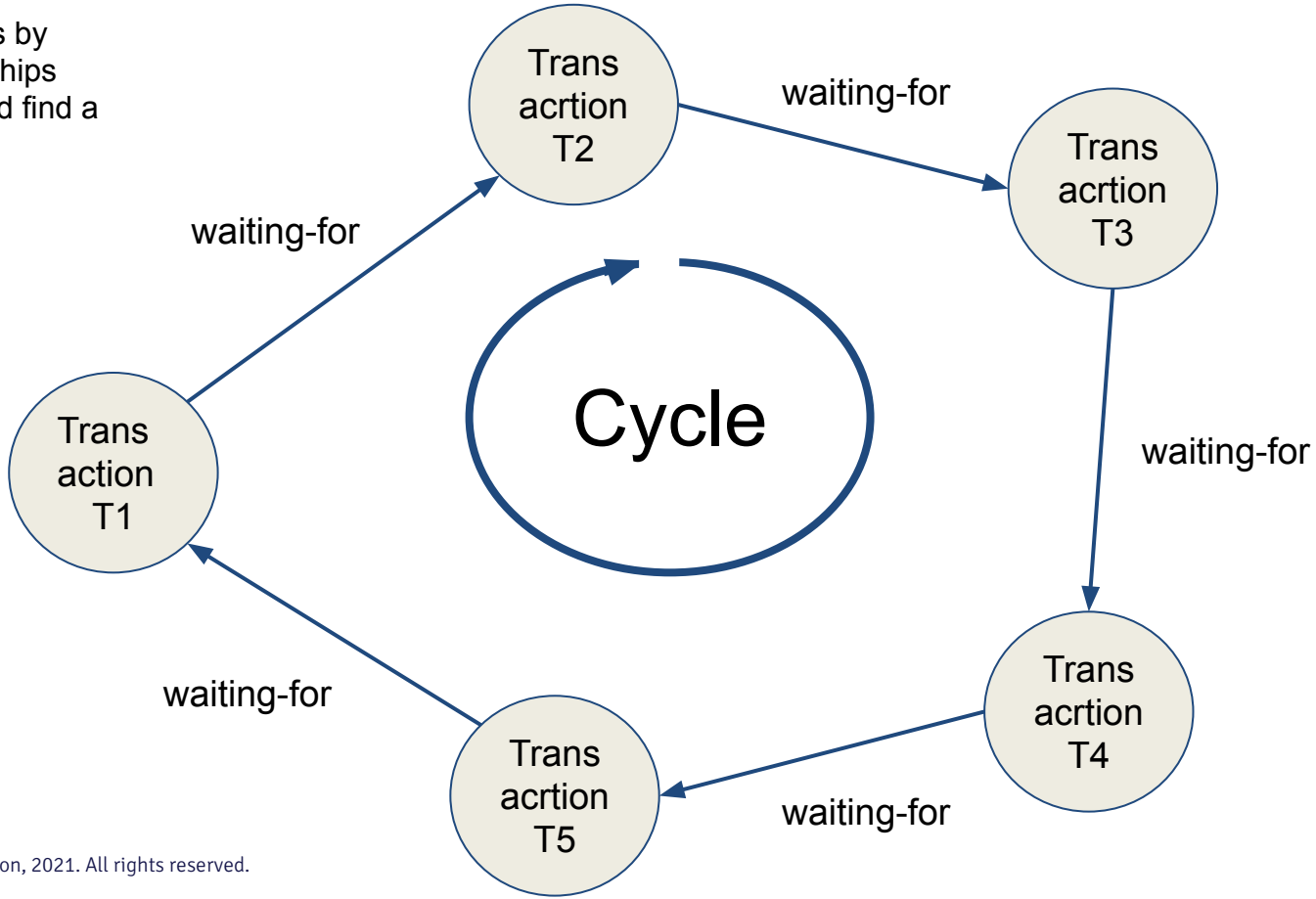
- Deadlock detection is to find a cycle in a wait-for-graph

Wait-for-graph for simple deadlock situation



Wait-for-graph for complicated deadlock situation

We can detect deadlocks by tracking waiting relationships between transactions and find a cycle.



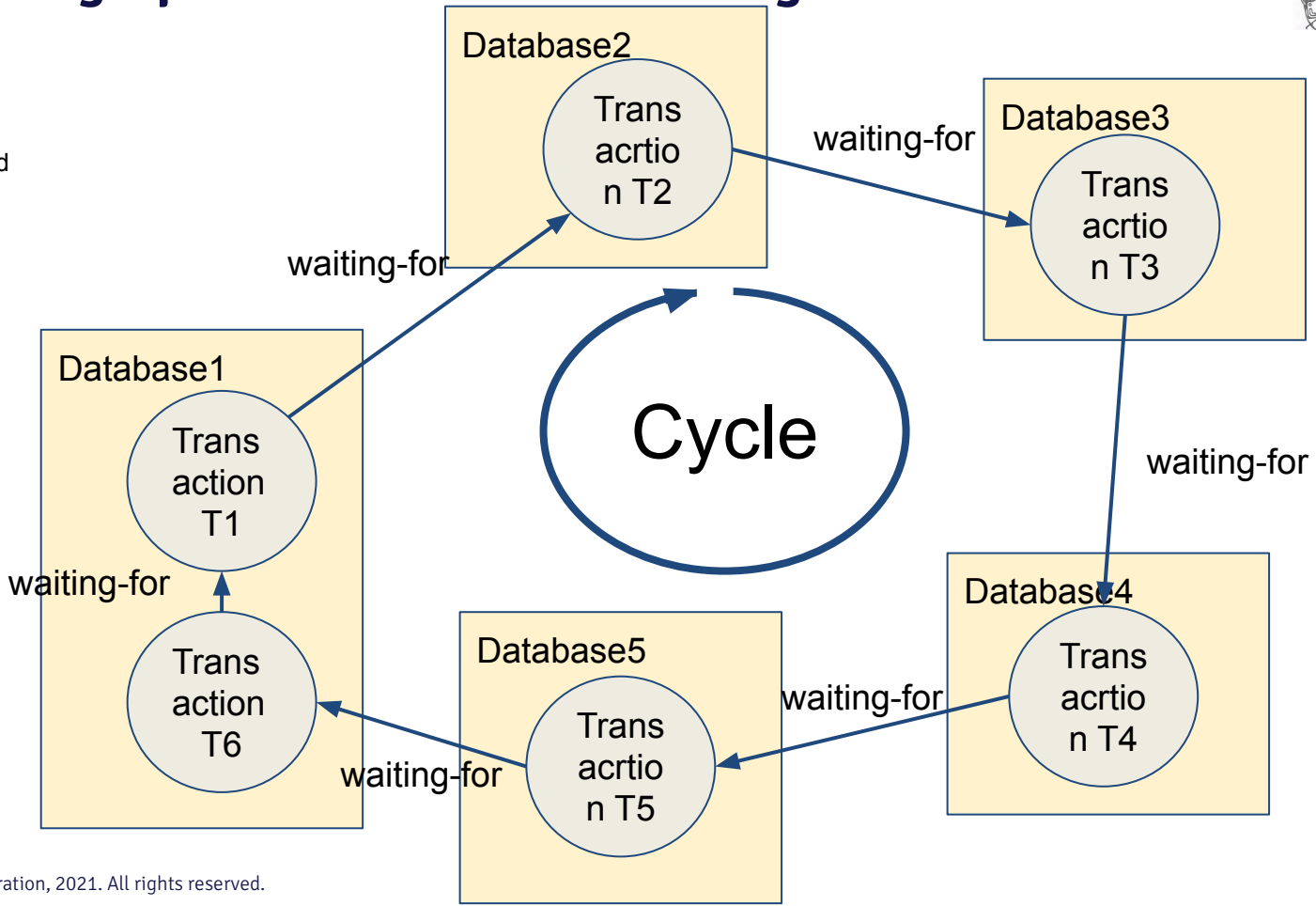
EDB

Global deadlock scenario

- If a transaction is waiting for remote transaction, we have global deadlock chance.
- There are no difference in wait-for-graph and cycle properties even in the global deadlock.

Wait-for-graph for transactions waiting for remote transaction

We can detect deadlocks by tracking waiting relationships between transactions and find a cycle. It is the same as local deadlock detection.



Lock system in PostgreSQL

PostgreSQL local deadlock detection is using this level of lock systenm

LOCK	Highest level lock for SQL level object. This level obeys the two-phase lock protocol. This level of the lock is involved in local deadlock detection.
LWLock	Light-weight lock.
PG_Semaphore	Portable semaphore implementation.
Spinlock	Short-duration data protection and synchronization
Latch	Synchronization

Use

Use

EDB

Deadlock Detection in PostgreSQL (1)

deadlock.c in src/backend/storage/lmgr

- If a transaction cannot acquire LOCK within a certain time (deadlock_timeout), deadlock detection is invoked (DeadLockCheck).
- Traces the lock a given transaction is waiting for (PGPROC, as in proc.h)
  - Waiting LOCK is in waitLock in PGPROC.
- If the transaction is waiting for a lock, deadlock detection traces all the transactions holding the lock.
  - Wait-for-graph is constructed this way.
- Repeat the above until wait-for-graph ends with a transaction waiting for nothing or a cycle is found

Deadlock Detection in PostgreSQL (2)

deadlock.c in src/backend/storage/lmgr

- If a cycle is found starting with the first transaction, this is regarded as DEADLOCK and the starting transaction aborts.
- If a cycle not starting with the first transaction is found, deadlock detector ignores it and regards as “NO DEADLOCK”. This cycle should be detected by other transaction.
- If no deadlock is found and the starting transaction has to wait, it changes the wait order of the lock for more chance of lock acquisition.

- Need to acquire low-level lock (LWLock) for all the lock objects.

EDB

Global deadlock detection requirements

- Wider usecase for various remote transaction dependency
  - FDW
  - BDR
  - Any other remote transaction use cases
- Compatible with different version of PostgreSQL
  - Different major versions in involved PostgreSQL instances
    - Maybe later than PG14
- Improve low-level locks acquisition
  - Release low-level locks while tracking remote transaction wait-for-graph
- Possibility to support other databases supported by FDW.

LOCK extension to represent remote transaction

New lock type EXTERNAL_LOCK to represent remote transaction which a transaction is waiting for.

Need more info other than simple LOCK:

- Connection string to remote database,
- Transaction ID of the remote transaction,
- Process information about the remote transaction, used to determine if detected wait-for-graph cycle is stable and is actual deadlock.
  - PID and transaction ID of the backend of the remote transaction,
    - Locigal XID to represent read transaction as well,
  - Index of PGPROC array (pgprocno),

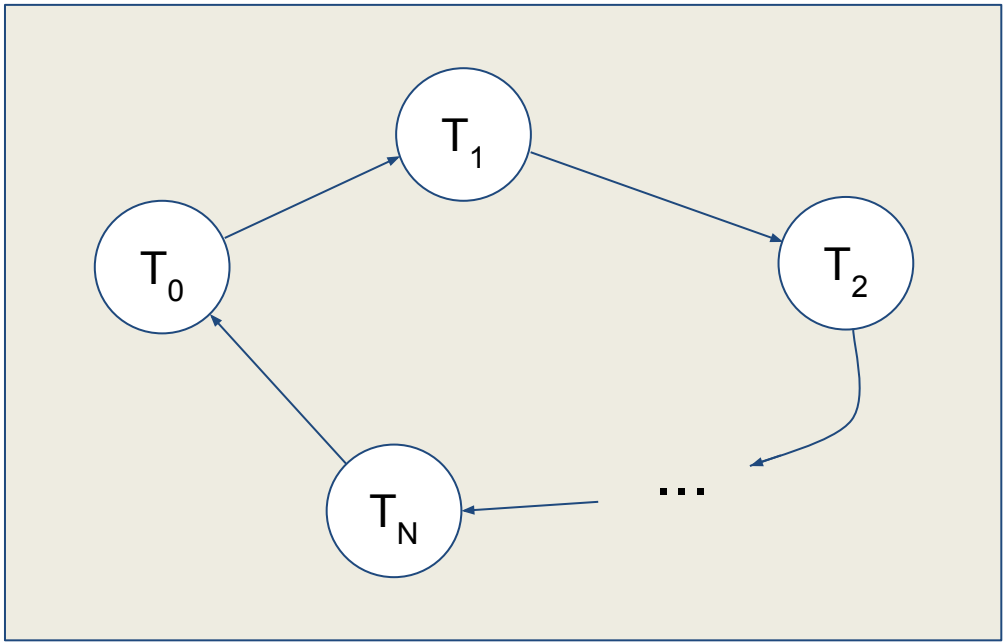
Other things to do:

- Lock holder is not in the local database, it's at remote:
  - Waiting transaction acquires the lock on behalf of the real lock holder,
  - The local transaction should both acquire and wait for the lock,
  - Additional information to track remote wait-for-graph.
- Dedicated internal API for these operations.
- Internally, LOCK API is used/extended to implement these additional properties.

Original feature of DeadLockCheck()

Denotes T_0 is waiting for T_1 to terminate

Find local wait-for-graph (WfG) cycle

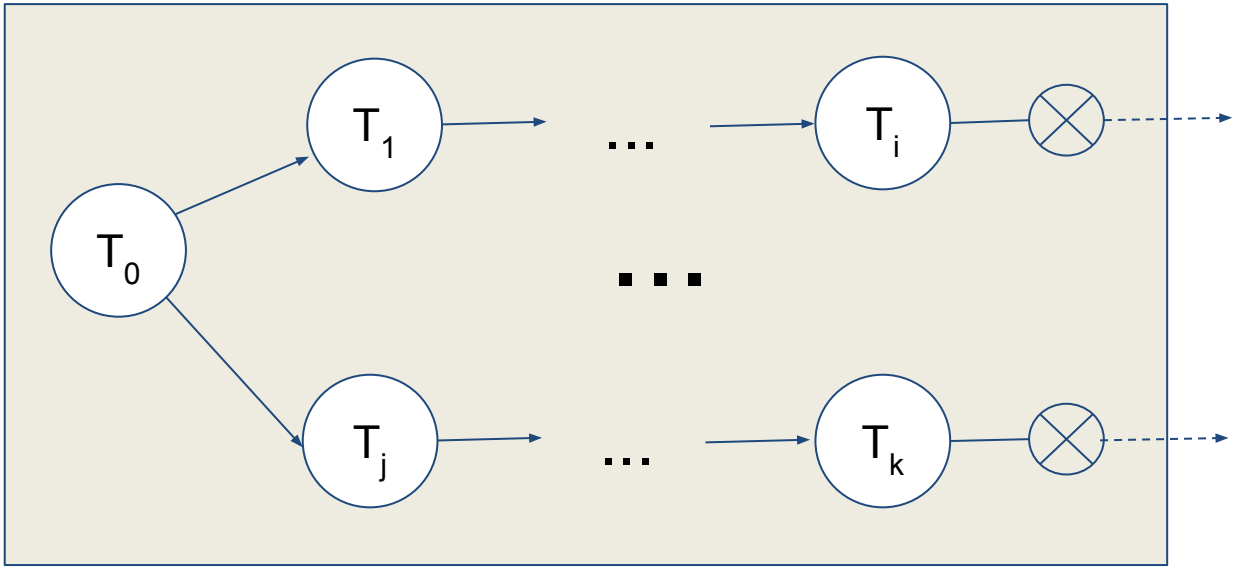


A lock exists between each transaction, which is waited by waiting transaction and acquired by waited transaction. This is omitted to simplify the figure.

Additional feature of DeadLockCheck()

Denotes T_0 is waiting for T_1 to terminate

External Lock (represents waiting for a remote transaction)



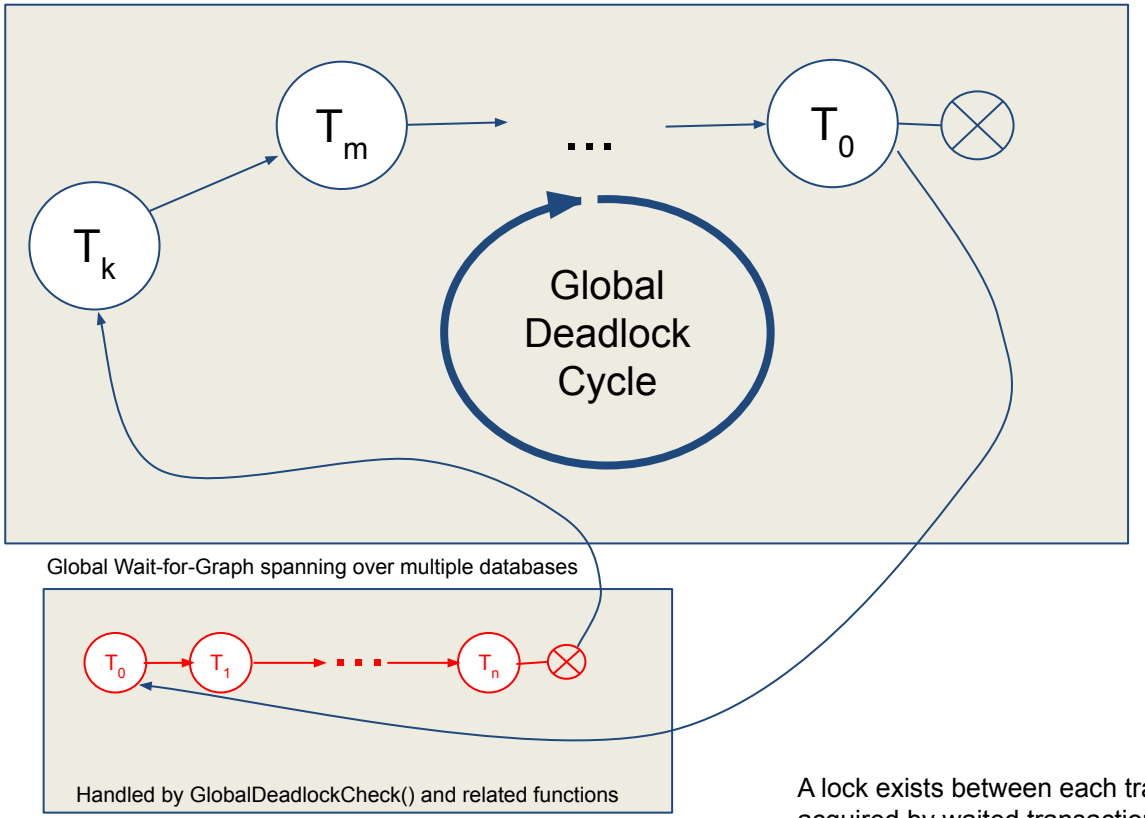
Waiting for a remote transaction

Waiting for a remote transaction

A lock exists between each transaction, which is waited by waiting transaction and acquired by waited transaction. This is omitted to simplify the figure.

Additional feature of DeadLockCheck()

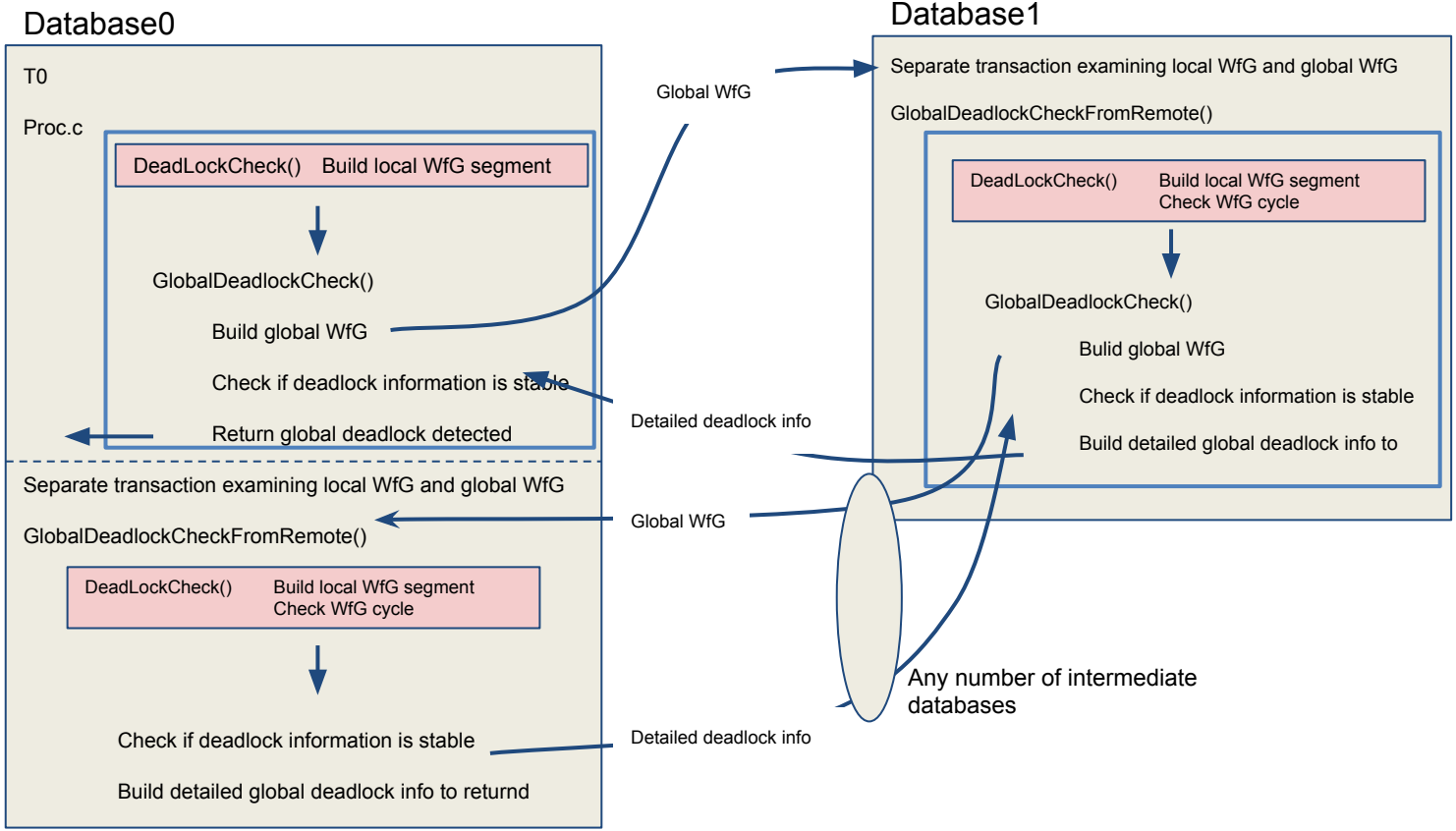
Determine if the backend is the origin of global WfG (global deadlock)



A lock exists between each transaction, which is waited by waiting transaction and acquired by waited transaction. This is omitted to simplify the figure.

GlobalDeadlockCheck() and related functions

Run with all LWLock acquired, based on current DeadLockCheck() implementation.



Extension to local deadlock detection

Detection of EXTERNAL_LOCK

- If a transaction is waiting for remote transaction, it should tell the lock subsystem that the backend is waiting for remote transaction.
  - API for extended lock.c
- A transaction (or its external worker) provides additional external lock information to the lock subsystem as shown in the previous slide.
- When lock acquisition cannot be made within deadlock_timeout, proc subsystem invokes DeadLockCheck().
  - When local deadlock is detected, the transaction is terminated, no change.
  - When EXTERNAL_LOCK is found in wait-for-graph, deadlock.c builds local wait-for-graph.
    - This local wait-for-graph is combined with additional external_lock information.
    - GlobalDeadlockCheck() invokes global deadlock check feature at the next database, transfer local wait-for-graph.
    - This process continues until
      - No further wait-for-graph is found, or
      - Wait-for-graph comes back to the original database and dead lock cycle is found, or
      - Further EXTERNAL_LOCK is found.

When wait-for-graph cycle is found

- Each database rechecks if local wait-for-graph is stable (*1)
  - Check if each PGPROC involved dealing with the same pid and same transaction
- If it is not stable
  - Examine other candidate wait-for-graphs
- If it is stable
  - Discovered wait-for-graph represents global deadlock.

(*1) If an wait-for-graph is a part of a deadlock, this wait-for graph should not change over time (such transaction should continue to wait for the same transaction)

What distributed transaction application should do

When it is going to wait for remote transaction

- Tell lock subsystem that it is waiting for EXTERNAL LOCK
  - `LockAcquireResult ExternalLockAcquire(PGPROC *proc, LOCKTAG *locktag)`
  - No need to specify the locktag. It is created and returned.
- Provide additional information for the external transaction
  - ```
    bool ExternalLockSetProperties(LOCKTAG *locktag,
                    PGPROC *proc, char *dsn, int target_pgprocno,
                    int target_pid, TransactionId target_xid, bool update_flag);
    ```
  - `target_pgprocno, target_pid, target_xid` are used to check stability of global wait-for-graph
  - They are stored in DataDir/pg_external_locks directory
- Tell the lock system that the process is waiting for external transaction using EXTERNAL LOCK
  - `bool ExternalLockWaitProc(const LOCKTAG *locktag, PGPROC *proc)`
- If connection to remote database is done in separate process/worker
  - Will provide SQL function/C function to take care of the above from clients.
    ```

Low level locks

- All LWLock will be held during the tracking of local wait-for-graph
  - Same as local deadlock check.
- When external lock is found and invoke remote wait-for-graph tracking
  - Invoke dedicated SQL function at the remote database with wait-for-graph discovered so far.
    - Remote database information and remote transaction information should be provided by remote transaction application (such as BDR or FDW-2PC) in advance.
  - All the local LWLocks are released
  - If global wait-for-graph is discovered
    - Check if local wait-for-graph is stable
    - If stable, it is a part of the global deadlock
    - If not, it is not a part of the global deadlock

Demonstration (1)

Run three transactions:

- TX1: at ksubuntu
  - Locks table T1
  - Waiting for TX2 running on ubuntu00
- TX2: at ubuntu00
  - Waiting for TX3 running on ksubuntu
- TX3: at ksubuntu
  - Attempt to lock table T1

The demonstration uses SQL functions for the test, essentially invoke additional lock functions for external lock. Detailed information will be provided if needed.

Tx1

```
koichi=# BEGIN;
BEGIN
koichi=# LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;
LOCK TABLE
koichi=# select * from gdd_test_external_lock_acquire_myself('a');
-[ RECORD 1 ]----------
label      | a
result     | LOCKACQUIRE_OK
field1     | 98
field2     | 314011
field3     | 6
field4     | 0
locktype   | LOCKTAG_EXTERNAL
lockmethod | 1

koichi=# select gdd_test_external_lock_set_properties_myself('a', 'host=ubuntu00 dbname=koichi user=koichi', 99, 32260, 74, true);
-[ RECORD 1 ]--------------------------------+--
gdd_test_external_lock_set_properties_myself | t

koichi=# select gdd_test_external_lock_wait_myself('a');
-[ RECORD 1 ]-----------------------+--
gdd_test_external_lock_wait_myself | t

koichi=#
```

Lock the table

Acquire external lock
Lock has a label (just works for the test function) for test convenience

Supply additional info for the external lock
waiting for Tx2 at ubuntu00

Begin waiting for the external lock

Tx2

```
koichi=# BEGIN;
BEGIN
koichi=# select * from gdd_test_external_lock_acquire_myself('b');
-[ RECORD 1 ]----------
label      | b
result     | LOCKACQUIRE_OK
field1     | 99
field2     | 32260
field3     | 74
field4     | 0
locktype   | LOCKTAG_EXTERNAL
lockmethod | 1
```

Acquire external lock
Lock has a label (just works for the test function) for test convenience

Supply additional info for the external lock
waiting for Tx3 at ksubuntu

```
koichi=# select gdd_test_external_lock_set_properties_myself('b', 'host=ksubuntu dbname=koichi user=koichi', 99, 313946, 81, true);
-[ RECORD 1 ]--------------------------------+--
gdd_test_external_lock_set_properties_myself | t

koichi=# select gdd_test_external_lock_wait_myself('b');
-[ RECORD 1 ]-----------------------+--
gdd_test_external_lock_wait_myself | t

koichi=#
```

Begin waiting for the external lock

Tx3

Issue SQL to cause global deadlock

```
koichi=# BEGIN;
BEGIN
koichi=# LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;
ERROR:  global deadlock detected
DETAIL:
Deadlock info for Database system id: 5f603ebbaa105997,  Process 313946 waits for AccessExclusiveLock on relation 16418 of
database 16384; blocked by process 314011.  Process 314011 waits for remote database 5f603ebd13b48d57, process 32260.
Deadlock info for Database system id: 5f603ebd13b48d57,  Process 32260 waits for remote database 5f603ebbaa105997, process
313946.
Deadlock info for Database system id: 5f603ebbaa105997,  Process 313946 waits for AccessExclusiveLock on relation 16418 of
database 16384; blocked by process 22026.
HINT:  See server log for query details.
koichi=#
```

LOG excerpt at ksubuntu (Tx1 and Tx3 running)

Wait-for-graph from ksubuntu to ubuntu00d

```
2020-09-15 13:13:07.662 JST [313946] DEBUG:  Executing worker "pg_gdd_check_worker c 'host=ubuntu00 dbname=koichi user=koichi'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_313946.in'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_313946.out'", Input data: "( 80800001 ( 1 ( 80880001 5f603ebbaa105997
00000003 ( 313946 99 81 ) ( 2 ( ( 16384 16418 0 0 0 1 ) 8 313946 99 81 ) ( ( 98 314011 6 0 8 1 ) 8 314011 98 6 ) ) (  'LOCK TABLE t1 IN ACCESS
EXCLUSIVE MODE;' 'select * from gdd_test_show_myself();' ) ( 314011 98 6 0 'host=ubuntu00 dbname=koichi user=koichi' 32260 99 74 ) ) ) )"
```

Wait-for-graph returned from ubuntu00

```
2020-09-15 13:13:07.662 JST [313946] DEBUG:  Issuing worker command: pg_gdd_check_worker c 'host=ubuntu00 dbname=koichi user=koichi'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_313946.in'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_313946.out'
Downstream database backend pid: 32263
2020-09-15 13:13:07.671 JST [313946] DEBUG:  Worker output: "1
    3, ( 80800001 ( 3 ( 80880001 5f603ebbaa105997 00000003 ( 313946 99 81 ) ( 2 ( ( 16384 16418 0 0 0 1 ) 8 313946 99 81 ) ( ( 98 314011 6 0 8 1
) 8 314011 98 6 ) ) (  'LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;' 'select * from gdd_test_show_myself();' ) ( 314011 98 6 0 'host=ubuntu00
dbname=koichi user=koichi' 32260 99 74 ) )( 80880001 5f603ebd13b48d57 00000003 ( 32260 99 74 ) ( 1 ( ( 99 32260 74 0 8 1 ) 8 32260 99 74 ) ) (
'select * from gdd_test_show_myself();' ) ( 32260 99 74 0 'host=ksubuntu dbname=koichi user=koichi' 313946 99 81 ) )( 80880001
5f603ebbaa105997 00000000 ( 1 ( ( 16384 16418 0 0 0 1 ) 8 313946 99 81 ) ) (  'LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;' )  ) ) )
    "
```

Global deadlock detailed info

```
2020-09-15 13:13:07.672 JST [313946] ERROR:  gobal deadlock detected
2020-09-15 13:13:07.672 JST [313946] DETAIL:
    Deadlock info for Database system id: 5f603ebbaa105997,  Process 313946 waits for AccessExclusiveLock on relation 16418 of database 16384;
blocked by process 314011.  Process 314011 waits for remote database 5f603ebd13b48d57, process 32260.  Process 313946: "LOCK TABLE t1 IN
ACCESS EXCLUSIVE MODE;" Process 314011: "select * from gdd_test_show_myself();"
    Deadlock info for Database system id: 5f603ebd13b48d57,  Process 32260 waits for remote database 5f603ebbaa105997, process 313946.
Process 32260: "select * from gdd_test_show_myself();"
    Deadlock info for Database system id: 5f603ebbaa105997,  Process 313946 waits for AccessExclusiveLock on relation 16418 of database 16384;
blocked by process 22026.  Process 313946: "LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;"
2020-09-15 13:13:07.672 JST [313946] HINT:  See server log for query details.
2020-09-15 13:13:07.672 JST [313946] STATEMENT:  LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;
```

LOG excerpt at ubuntu00 (Tx2 is running)

Wait-for-graph from ubuntu00 to ksubuntu

2020-09-15 13:13:07.670 JST [32263] DEBUG: Executing worker "pg_gdd_check_worker c 'host=[illegible] dbname=koichi user=koichi'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_32263.in'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_32263.out'", Input data: "(80800001 (2 (80880001 5f603ebbaa105997 00000003 (313946 99 81) (2 ((16384 16418 0 0 0 1) 8 313946 99 81) ((98 314011 6 0 8 1) 8 314011 98 6)) ('LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;' 'select * from gdd_test_show_myself();') (314011 98 6 0 'host=ubuntu00 dbname=koichi user=koichi' 32260 99 74))(80880001 5f603ebd13b48d57 00000003 (32260 99 74) (1 ((99 32260 74 0 8 1) 8 32260 99 74)) ('select * from gdd_test_show_myself();') (32260 99 74 0 'host=ksubuntu dbname=koichi user=koichi' 313946 99 81))))"
2020-09-15 13:13:07.670 JST [32263] DEBUG: Issuing worker command: pg_gdd_check_worker c 'host=ksubuntu dbname=koichi user=koichi'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_32263.in'
'/home/koichi/gdd_test/pg12_gdd_database/pg_external_locks/wfg_32263.out'
Downstream database backend pid: 314044
2020-09-15 13:13:07.674 JST [32263] DEBUG: Worker output: "1
    3, (80800001 (3 (80880001 5f603ebbaa105997 00000003 (313946 99 81) (2 ((16384 16418 0 0 0 1) 8 313946 99 81) ((98 314011 6 0 8 1) 8 314011 98 6)) ('LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;' 'select * from gdd_test_show_myself();') (314011 98 6 0 'host=ubuntu00 dbname=koichi user=koichi' 32260 99 74))(80880001 5f603ebd13b48d57 00000003 (32260 99 74) (1 ((99 32260 74 0 8 1) 8 32260 99 74)) ('select * from gdd_test_show_myself();') (32260 99 74 0 'host=ksubuntu dbname=koichi user=koichi' 313946 99 81))(80880001 5f603ebbaa105997 00000000 (1 ((16384 16418 0 0 0 1) 8 313946 99 81)) ('LOCK TABLE t1 IN ACCESS EXCLUSIVE MODE;'))))
    "

Wait-for-graph returned from ksubuntu

Resources

Source repo

```
https://github.com/koichi-szk/postgres.git
```

Test tools/environment

```
https://github.com/koichi-szk/gdd_test.git
```

They are now private. Please let me know if you are interested in.

Thank you very much

Koichi Suzuki

koichi.suzuki@enterprisedb.com