PostgreSQL Database Locks

Last updated by | Hamza Aqel | Jan 25, 2022 at 2:03 AM PST

Please don't modify or move as this is part of GT , please contact haaqel@microsoft.com if needed

For any OLTP applications, there is a possibility that the customer application will face a row level locking and that will impact his query execution or application functionality in terms of Latency and unexpected behavior , the key player in investigating such issue is from the application/customer side as this is an application behavior , here are some steps that you can use in checking if there is any DB locking :

From our telemetry (Kusto):

You will see something like this in MonRdmsPgSqlSandbox :

```
let TimeCheckStart = datetime('2020-07-22T05:30:00');

let TimeCheckEnd = datetime('2020-07-22T06:20:00');

MonRdmsPgSqlSandbox

| where LogicalServerName contains "server-name-here"

| where TimeCheckStart < TIMESTAMP and TIMESTAMP < TimeCheckEnd

| where text  contains "of relation"

|project originalEventTimestamp,text
```

You will see output in case there are DB locks similar to :

```
2020-07-22 05:36:48.5151335 2020-07-22 05:36:48 UTC-5f17c4db.a2d5c-LOG:  process 666972 still waiting for ExclusiveLock on tuple (92546,36) of relation 27166 of database 27016 after 1015.641 ms

2020-07-22 05:36:48.7182451 2020-07-22 05:36:48 UTC-5f17c4db.a2d5c-LOG:  process 666972 acquired ExclusiveLock on tuple (92546,36) of relation 27166 of database 27016 after 1218.801 ms

2020-07-22 05:49:18.7464581 2020-07-22 05:49:18 UTC-5f17ce00.a30d8-LOG:  process 667864 still waiting for ExclusiveLock on tuple (92561,35) of relation 27166 of database 27016 after 1000.006 ms

2020-07-22 05:49:19.6961496 2020-07-22 05:49:19 UTC-5f17ce00.a30d8-LOG:  process 667864 acquired ExclusiveLock on tuple (92561,35) of relation 27166 of database 27016 after 1937.476 ms
```

```
2020-07-22 05:50:05.8723324 2020-07-22 05:50:05 UTC-5f17ca7f.a2f48-LOG:  process 667464 still waiting for ExclusiveLock on tuple (92510,35) of relation 27166 of database 27016 after 1015.579 ms

2020-07-22 05:50:07.3876439 2020-07-22 05:50:07 UTC-5f17d36c.a3aec-LOG:  process 670444 still waiting for ExclusiveLock on tuple (92510,35) of relation 27166 of database 27016 after 1015.593 ms

2020-07-22 05:50:11.4267184 2020-07-22 05:50:11 UTC-5f17ca7f.a2f48-LOG:  process 667464 acquired ExclusiveLock on tuple (92510,35) of relation 27166 of database 27016 after 6562.455 ms
```

From Application/Customer Side:

Locking informationand DMvs to investigate
https://www.citusdata.com/blog/2018/02/15/when-postgresql-blocks/

When Postgres blocks: 7 tips for dealing with locks
https://www.citusdata.com/blog/2018/02/22/seven-tips-for-dealing-with-postgres-locks/

And here are some useful queries that you can check with the customer:

https://wiki.postgresql.org/wiki/Lock_Monitoring

Combination of blocked and blocking activity

The following query may be helpful to see what processes are blocking SQL statements (these only find row-level locks, not object-level locks).

```
SELECT blocked_locks.pid     AS blocked_pid,
        blocked_activity.usename  AS blocked_user,
        blocking_locks.pid     AS blocking_pid,
        blocking_activity.usename AS blocking_user,
        blocked_activity.query    AS blocked_statement,
        blocking_activity.query   AS current_statement_in_blocking_process
  FROM  pg_catalog.pg_locks         blocked_locks
   JOIN pg_catalog.pg_stat_activity blocked_activity  ON blocked_activity.pid = blocked_locks.pid
   JOIN pg_catalog.pg_locks         blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
    JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = blocking_locks.pid
  WHERE NOT blocked_locks.GRANTED;
```

Here's an alternate view of that same data that includes application_name's

Setting application_name variable in the begging of each transaction allows you to which logical process blocks another one. It can be information which source code line starts transaction or any other information that helps you to match application_name to your code.

```
SET application_name='%your_logical_name%';
SELECT blocked_locks.pid     AS blocked_pid,
        blocked_activity.usename  AS blocked_user,
        blocking_locks.pid     AS blocking_pid,
        blocking_activity.usename AS blocking_user,
        blocked_activity.query    AS blocked_statement,
        blocking_activity.query   AS current_statement_in_blocking_process,
        blocked_activity.application_name AS blocked_application,
        blocking_activity.application_name AS blocking_application
   FROM  pg_catalog.pg_locks         blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity  ON blocked_activity.pid = blocked_locks.pid
    JOIN pg_catalog.pg_locks         blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
     JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

Note: While this query will mostly work fine, it still has some correctness issues [1], particularly on 9.6.

Here's an alternate view of that same data that includes an idea how old the state is

```
  SELECT a.datname,
        c.relname,
        l.transactionid,
        l.mode,
        l.GRANTED,
        a.usename,
        a.current_query,
        a.query_start,
        age(now(), a.query_start) AS "age",
        a.procpid
    FROM  pg_stat_activity a
     JOIN pg_locks         l ON l.pid = a.procpid
     JOIN pg_class         c ON c.oid = l.relation
    ORDER BY a.query_start;
```

From <https://wiki.postgresql.org/wiki/Lock_Monitoring>

Created with Microsoft OneNote 2016.

How good have you found this content?