Lock Request Timeout Error 1222 State 112

Last updated by | Holger Linke | Oct 26, 2022 at 1:41 AM PDT

Contents

Issue

Users receiving lock timeout request (Error 1222, State 112).

Investigation/Analysis

Lock timeout requests error 1222 with state 112 indicates Query Data Store (QDS) is unable to acquire X locks on a statement (and/or an execution plan) to store/retrieve data from the query store. QDS sets INFINITE as the default value for the lock timeout request thus users should not be seeing Error 1222 with state 112. However, there is a code path that the QDS process can possibly go through which results in the timeout value being set to other than INFINITE, and this is how users are seeing this error (Details at the end of this TSG).

Error 1222 with state 112 is raised at AcquireQdsStatementLock method. The constant ABORT_QDS_LOCK_STMT is defined as 112 which appears as the state number in the error message.

```
void CQDSTransactionLockProxy::AcquireQdsStatementLock(…, lockTimeout)
…
…
 if (LCK_FAILED(result))
    {
        lck_StandardHandler(result, ABORT_QDS_LOCK_STMT);
    }
```

Mitigation

In most cases this issue can be mitigated by turning off the QueryStoreStatementHints feature switch (FS). In this case you can open an ICM and have the FS turn off.

Following are check items to determine whether turning off the FS will mitigate the issue.

- Check the current status of the feature switch.

```
MonConfigLogicalServerOverrides
| where Logical_Server_Name =~ "{LogicalServerName}"
| where Property_Name has "QueryStoreStatementHints"
| project-reorder Property_Name,  Property_Value, Logical_Server_Name
| summarize by Property_Name,  Property_Value, Logical_Server_Name, TIMESTAMP
```

Property_Value of ON indicates the feature switch is enabled. The above Kusto query might not return any result if a CAS command (Set-ServerConfigurationParameters) had never been issued on the customer's server. In that case, you can assume the FS is ON (need to double check with PG).

- Check the statement operation types executed surrounding the error encountered:

```
MonWiQdsExecStats
| where TIMESTAMP between (datetime({StartTime})..datetime({EndTime}))
| where AppName =~ "{AppName}"
| where LogicalServerName =~ "{LogicalServerName}"
| where database_name =~ "{LogicalDatabaseName}"
| where statement_type == "x_estypInsertBulk"
| project TIMESTAMP, originalEventTimestamp, event, statement_type
| limit 1000
```

Based on an analysis of ICM cases related to this issue, it is likely that this issue will arise when the user is performing bulk inserts, specifically through an application that uses the SqlClient class, SqlBulkCopy. However, this does not mean that other operations (SELECT, INSERT, etc.) will not trigger this issue. If there were many bulk inserts performed prior to the lock timeout error, it is very likely that turning off the FS can narrow down the problem. (Further research to be continued)

RCA Template (optional)

(TBD)

More Information (Detailed Analysis)

This section describes the findings based on source code analysis to understand how the lock timeout error with state 112 is raised.
** Note the analysis is based as of Jun, 2021, thus partial code listed here may differ going further.

As mentioned previously, error 1222, state 112 is raise in AcquireQdsStatementLock method. Going through code we've found there are four places that make calls to this method.

- CQueryStoreQuerySTVF::InternalGetRow
- CQueryStorePlanSTVF::InternalGetRow
- CQDSTransactionLockProxy::AcquireStatementLock(UINT32 stmtKeyHash, EQDSLockMode lockMode, DWORD lockTimeout)
- CQDSTransactionLockProxy::AcquirePSPQueryVariantStatementLock

The first three methods pass INFINITE as the lockTimeout parameter for AcquireQdsStatementLock, hence the issue should not occur from these calls.

CQueryStoreQuerySTVF::InternalGetRow

```
HRESULT CQueryStoreQuerySTVF::InternalGetRow()
…
…
    if (NULL != m_apCurrentQdsQuery)
        {
        // Acquire stability lock for the query
        CQDSTransactionLockProxy::AcquireQdsStatementLock(m_pXdes,
            m_dbid,
            m_apCurrentQdsQuery->GetStatementKeyHashForItem(),
            lm_Shared,
            INFINITE /* lockTimeout */);
```

CQueryStorePlanSTVF::InternalGetRow

```
HRESULT CQueryStorePlanSTVF::InternalGetRow()
if (NULL != m_apCurrentQdsPlan)
        {
            // Acquire stability lock for the plan
            CQDSTransactionLockProxy::AcquireQdsStatementLock(m_pXdes,
                m_dbid,
                m_apCurrentQdsPlan->GetStatementKeyHashForItem(),
                lm_Shared,
                INFINITE /* lockTimeout */);
```

CQDSTransactionLockProxy::AcquireStatementLock AcquireStatementLock is called by these method. All methods pass INFINITE as the parameter for the lockTimeout value, hence, it is unlikely that timeout error will occur from these calls.

- CDBQDS::CleanupStaleQuery
- CDBQDS::UpdateContextSettingsId
- GetStatementExclusiveLock
- GetStatementSharedLock

- Who's calling CQDSTransactionLockProxy::AcquirePSPQueryVariantStatementLock?

The following partial code shows AcquirePSPQueryVariantStatementLock is being called by GetPSPQueryVariantStatementExclusiveLock with the lockTimeout parameter set to INFINITE. But since the INFINITE value is an optional parameter here we further investigate to find out who is calling GetPSPQueryVariantStatementExclusiveLock and what parameter is being passed to this method.

```
 // We don't need a shared version of this as there is no code path
    // that not needing exclusive lock for the query variant.
    inline void GetPSPQueryVariantStatementExclusiveLock(
        __in IQDSTransactionLockProxy *pTranLockProxy,
        __in UINT32                   parentQueryHash,
        __in UINT32                   queryVariantHash,
        __in DWORD                    lockTimeout = INFINITE)
    {
        pTranLockProxy->AcquirePSPQueryVariantStatementLock(
            parentQueryHash, queryVariantHash, lm_Exclusive, lockTimeout);
    }
```

- Who's calling GetPSPQueryVariantStatementExclusiveLock?

Here we see that GetPSPQueryVariantStatementExclusiveLock is being called with setting the lockTimeout parameter with a call to GetLckTimeoutLimit method.

```
void CDBQDS::StoreDispatcherStmtAndPlanForPSP
…
…
    // Once we have the parent statement locked, there is guarantee that this
    // query variant stmt will get the lock
    GetPSPQueryVariantStatementExclusiveLock
    (
        apLockProxy,
        pParentQueryKey->UiHashCode(),
        pQueryVariantKey->UiHashCode(),
        PGetQDSManagerInternal()->GetLckTimeoutLimit()
```

- Definition of PGetQDSManagerInternal()->GetLckTimeoutLimit()

```
// Returns the lock timeout for acquiring Query Store locks on query compile and execution path
   DWORD GetLckTimeoutLimit() const
   {return m_lckTimeoutLimitMilliseconds; }
```

- Where does the timeout value get set?

```
void CQDSManager::RefreshQDSSettings()
{
        PGetQDSHost()->LoadDWORDConfigValueForQdsSetting(
                XDB_CONFIG_QDS_LIMITED_LCK_TIMEOUT_MILLISECONDS,
                m_lckTimeoutLimitMilliseconds,
                x_lckTimeoutLimitMilliseconds);
```

- Where is XDB_CONFIG_QDS_LIMITED_LCK_TIMEOUT_MILLISECOND defined

Related file	Definition
xdbsrvglobals.h	#define XDB_CONFIG_QDS_LIMITED_LCK_TIMEOUT_MILLISECOND L"LimitedLckTimeoutMilliseconds"
ServiceSettings_SQLServer_Common.xml	`<Setting name="LimitedLckTimeoutMilliseconds" introducedAt="2015-09-16" dataType="int" scope="applicationType"> <Description> Lock timeout for acquiring Query Store locks on query compile and execution path, where otherwise waiting on acquiring locks infinitely would block user queries. </Description> <Value>1000</Value> </Setting>`

- Further research to be continued

The next big question is what are the operations or conditions that triggers call to AcquirePSPQueryVariantStatementLock. This is still under research but from the name of the method, it might have something to do with a particular type of statement that QDS attempts to acquire a lock. As a trivia, the "PSP" that is part of the method name stands for "parameter-sensitive plan", thus the query statement that triggers the call to AcquirePSPQueryVariantStatementLock might have something to do with parametrize queries, but again this is still under research.

Public Doc Reference

How Query Store collects data

Query Store hints (Preview)

Internal Reference

ICM 237467095, 237436129, 235856081, 235736272, 235718382

Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause:
Performance/Waits/Other

How good have you found this content?

🙂 🙁