Wait type LCK_M_RS_U

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

Contents

Issue

The customer reports slowness in their application and the performance troubleshooting shows high blocking time on wait type `LCK_M_RS_U`.

For steps to investigate blocking, see Wiki article Blocking.

Investigation / Analysis

Possible causes

`LCK_M_RS_U` occurs when a task is waiting to acquire two locks: an Update lock on the current key value and an Update Range lock between the current and the previous key. The RangeS_U lock is held in two scenarios:

- Using the SERIALIZABLE isolation level, and/or:
- The blocked table has a unique nonclustered index with IGNORE_DUP_KEY = ON

The `IGNORE_DUP_KEY` will allow the insert of duplicate keys into a unique non-clustered index with warnings; it impacts performance and should be used carefully.

Troubleshooting

Use the following steps to troubleshoot this scenario:

(1) Use ASC --> SQL Troubleshooter --> Performance tab to check the overall performance insights and blocking status.

(2) Use the steps from the Wiki Blocking article to confirm the results if there is any further doubt.

(3) After verifying that the blocking is mainly caused by wait type `LCK_M_RS_U`, check with the customer if they have the SERIALIZABLE isolation level enabled for the query.

You can also verify this in the Kusto telemetry (query looks frightening but is powerful):

```
let startTime = datetime(2022-10-20 07:00:00Z);
let endTime = datetime(2022-10-27 12:00:00Z);
let srv = "servername";
let db = "databasename";
let blockingchain=MonBlockedProcessReportFiltered
| where TIMESTAMP > startTime
| where TIMESTAMP < endTime
| where LogicalServerName =~ srv
| where logical_database_name  =~ db
| where lock_mode =="RS_U"
//| where AppName =~ "a4b395401f2d" //and NodeName =~"DB.87"
| extend monitorLoop = extract("monitorLoop=\"([0-9]+)\"", 1, blocked_process_filtered, typeof(int))
| parse blocked_process_filtered with anystr3:string "<blocked-process>" blockee "</blocked-process>" discard1
| parse blocked_process_filtered with anystr4:string"<blocking-process>" blocker "</blocking-process>" discard
| extend blockee_session_id = extract("spid=\"([0-9]+)\"", 1, blockee, typeof(int))
| extend blockee_status = extract("status=\"(.*?)\"", 1, blockee, typeof(string))
| extend blockee_waittime = extract("waittime=\"([0-9]+)\"", 1, blockee, typeof(int))
| extend blockee_trancount = extract("trancount=\"([0-9]+)\"", 1, blockee, typeof(int))
| extend blockee_lasttranstarted = extract("lasttranstarted=\"(.*?)\"", 1, blockee, typeof(datetime))
| extend blockee_queryhash = extract("queryhash=\"(.*?)\"", 1, blockee, typeof(string))
| extend blockee_isolationlevel = extract("isolationlevel=\"(.*?)\"", 1, blockee, typeof(string))
| extend blockee_lastattention = extract("lastattention=\"(.*?)\"", 1, blockee, typeof(datetime))
| extend blockee_lastattention = extract("lastattention=\"(.*?)\"", 1, blockee, typeof(datetime))
| extend blockee_lastbatchstarted = extract("lastbatchstarted=\"(.*?)\"", 1, blockee, typeof(datetime))
| extend blockee_lastbatchcompleted = extract("lastbatchcompleted=\"(.*?)\"", 1, blockee, typeof(datetime))
| extend blockee_waitresource = extract("waitresource=\"(.*?)\"", 1, blockee, typeof(string))
| extend blockee_lockMode = extract("lockMode=\"(.*?)\"", 1, blockee, typeof(string))
| extend blockee_clientapp = extract("clientapp=\"(.*?)\"", 1, blockee, typeof(string))
| extend blocker_session_id = extract("spid=\"([0-9]+)\"", 1, blocker, typeof(int))
| extend blocker_status = extract("status=\"(.*?)\"", 1, blocker, typeof(string))
| extend blocker_waittime = extract("waittime=\"([0-9]+)\"", 1, blocker, typeof(int))
| extend blocker_trancount = extract("trancount=\"([0-9]+)\"", 1, blocker, typeof(int))
| extend blocker_queryhhash = extract("queryhash=\"(.*?)\"", 1, blocker, typeof(string))
| extend blocker_isolationlevel = extract("isolationlevel=\"(.*?)\"", 1, blocker, typeof(string))
| extend blocker_lastattention = extract("lastattention=\"(.*?)\"", 1, blocker, typeof(datetime))
| extend blocker_lastattention = extract("lastattention=\"(.*?)\"", 1, blocker, typeof(datetime))
| extend blocker_lastbatchstarted = extract("lastbatchstarted=\"(.*?)\"", 1, blocker, typeof(datetime))
| extend blocker_lastbatchcompleted = extract("lastbatchcompleted=\"(.*?)\"", 1, blocker, typeof(datetime))
| extend blocker_clientapp = extract("clientapp=\"(.*?)\"", 1, blocker, typeof(string))
//| project PreciseTimeStamp, monitorLoop, blockee_session_id , blocker_session_id
| order by monitorLoop asc nulls last
| limit 100
;
let leadblockers=
blockingchain
| join kind= rightanti (
   blockingchain
) on $left.monitorLoop == $right.monitorLoop and $left.blockee_session_id==$right.blocker_session_id
| extend lead_blocker_session_id = blocker_session_id
| distinct monitorLoop, lead_blocker_session_id;
leadblockers
| join kind= inner (
    blockingchain
) on $left.monitorLoop == $right.monitorLoop and $left.lead_blocker_session_id==$right.blocker_session_id
| order by TIMESTAMP asc nulls last
| project monitorLoop, originalEventTimestamp, AppName, LogicalServerName, ResourcePoolName, database_id, data
blocker_session_id, blocker_isolationlevel, blocker_queryhhash=toupper(blocker_queryhhash), blocker_status, bl
blockee_session_id, blockee_waitresource, blockee_lockMode, blockee_isolationlevel, blockee_queryhhash=toupper
//, blocked_process_filtered
```

Sample output:

monit...	originalEventTimesta...	AppName	Logi...	database_id	dat...	lead_blocker_...	duration	resource_owner_...	lock_mode	trans...	blocker_session...	blocker_isolationlevel	blocker_queryhhash	blocker_status	blocker_trancount
72044	2022-10-27 06:59:58....	b6dddbc...	pcd...	8	SIN...	2184	56799000	LOCK	RS_U	2869...	2184	read committed (2)	0XF964BCCD4C7A75FF	suspended	1
72044	2022-10-27 06:59:58....	b6dddbc...	pcd...	8	SIN...	2184	56796000	LOCK	RS_U	2869...	2184	read committed (2)	0XF964BCCD4C7A75FF	suspended	1
72044	2022-10-27 06:59:58....	b6dddbc...	pcd...	8	SIN...	2184	56814000	LOCK	RS_U	2869...	2184	read committed (2)	0XF964BCCD4C7A75FF	suspended	1
72053	2022-10-27 07:00:18....	b6dddbc...	pcd...	8	SIN...	2184	65426000	LOCK	RS_U	2870...	2184	read committed (2)	0XF964BCCD4C7A75FF	running	1
72053	2022-10-27 07:00:18....	b6dddbc...	pcd...	8	SIN...	2184	65419000	LOCK	RS_U	2870...	2184	read committed (2)	0XF964BCCD4C7A75FF	runnable	1

(4) If the SERIALIZABLE isolation level is not used, collect below index information from the customer to check if they are using any index with IGNORE_DUP_KEY set to ON.

The output from the blocked process report has a column "blockee_waitresource" with an output similar to "KEY: 8:72057595326758912 (61c8946058e6)". It confirms that we are indeed waiting on a "key" resource; the number behind the database ID ("8:" in this example) is the partition ID or HoBT ID ("72057595326758912" in this example). Take this partition/HoBT ID and fill it into the following SQL query:

```
SELECT
    o.name AS table_name,
    s.Name AS schema_name,
    i.index_id,
    i.type_desc AS index_type,
    i.name AS index_name,
    i.is_unique,
    i.ignore_dup_key
FROM
    sys.all_objects o
    INNER JOIN sys.indexes i ON o.OBJECT_ID = i.object_id
    INNER JOIN sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
    LEFT OUTER JOIN sys.schemas s ON o.schema_id = s.schema_id
WHERE
    p.hobt_id in (72057595326758912) -- hobt_id should be fetched from the blocking report
    OR (is_unique = 1 AND ignore_dup_key = 1)
ORDER BY o.Name, i.index_id;

/*
table_name      schema_name  index_id  index_type    index_name                          is_unique  ignore_dup_
BusinessEntity  Person       3         NONCLUSTERED  BusinessEntity_BusinessEntityID     1          1
*/
```

Mitigation

Disable the IGNORE_DUP_KEY property using following statement (this is for the table/index from the sample output above):

```
ALTER INDEX [BusinessEntity_BusinessEntityID] on Person.BusinessEntity SET ( IGNORE_DUP_KEY = OFF )
```

More Information

The following table from the Transaction locking and row versioning guide shows the resource lock modes that the SQL Server Database Engine uses:

Lock mode	Description
Shared (S)	Used for read operations that do not change or update data, such as a SELECT statement.
Update (U)	Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and potentially updating resources later.
Exclusive (X)	Used for data-modification operations, such as INSERT, UPDATE, or DELETE. Ensures that multiple updates cannot be made to the same resource at the same time.
Intent	Used to establish a lock hierarchy. The types of intent locks are: intent shared (IS), intent exclusive (IX), and shared with intent exclusive (SIX).
Schema	Used when an operation dependent on the schema of a table is executing. The types of schema locks are: schema modification (Sch-M) and schema stability (Sch-S).
Bulk Update (BU)	Used when bulk copying data into a table and the TABLOCK hint is specified.
Key-range	Protects the range of rows read by a query when using the serializable transaction isolation level. Ensures that other transactions cannot insert rows that would qualify for the queries of the serializable transaction if the queries were run again.

The wait type `LCK_M_RS_U` consists of several parts which are explained in [sys.dm_tran_locks (Transact-SQL)]. The relevant components would be `Sch-M - RangeS_U`:

- Sch-S (Schema stability) = Ensures that a schema element, such as a table or index, is not dropped while any session holds a schema stability lock on the schema element.
- **Sch-M (Schema modification) = Must be held by any session that wants to change the schema of the specified resource. Ensures that no other sessions are referencing the indicated object.**
- S (Shared) = The holding session is granted shared access to the resource.
- U (Update) = Indicates an update lock acquired on resources that may eventually be updated. It is used to prevent a common form of deadlock that occurs when multiple sessions lock resources for potential update in the future.
- X (Exclusive) = The holding session is granted exclusive access to the resource.
- IS (Intent Shared) = Indicates the intention to place S locks on some subordinate resource in the lock hierarchy.
- IU (Intent Update) = Indicates the intention to place U locks on some subordinate resource in the lock hierarchy.

- IX (Intent Exclusive) = Indicates the intention to place X locks on some subordinate resource in the lock hierarchy.
- SIU (Shared Intent Update) = Indicates shared access to a resource with the intent of acquiring update locks on subordinate resources in the lock hierarchy.
- SIX (Shared Intent Exclusive) = Indicates shared access to a resource with the intent of acquiring exclusive locks on subordinate resources in the lock hierarchy.
- UIX (Update Intent Exclusive) = Indicates an update lock hold on a resource with the intent of acquiring exclusive locks on subordinate resources in the lock hierarchy.
- BU = Used by bulk operations.
- RangeS_S (Shared Key-Range and Shared Resource lock) = Indicates serializable range scan.
- **RangeS_U (Shared Key-Range and Update Resource lock) = Indicates serializable update scan.**
- RangeI_N (Insert Key-Range and Null Resource lock) = Used to test ranges before inserting a new key into an index.
- RangeI_S = Key-Range Conversion lock, created by an overlap of RangeI_N and S locks.
- RangeI_U = Key-Range Conversion lock, created by an overlap of RangeI_N and U locks.
- RangeI_X = Key-Range Conversion lock, created by an overlap of RangeI_N and X locks.
- RangeX_S = Key-Range Conversion lock, created by an overlap of RangeI_N and RangeS_S. locks.
- RangeX_U = Key-Range Conversion lock, created by an overlap of RangeI_N and RangeS_U locks.
- RangeX_X (Exclusive Key-Range and Exclusive Resource lock) = This is a conversion lock used when updating a key in a range.

If a resource is already locked by another transaction, a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock. Lock compatibility controls whether multiple transactions can acquire locks on the same resource at the same time. If the mode of the requested lock is not compatible with the existing lock, the transaction requesting the new lock waits for the existing lock to be released. For a lock compatibility matrix, see Transaction locking and row versioning guide - Lock compatibility.

Public Doc Reference

- Maintaining Unique Indexes with IGNORE_DUP_KEY

How good have you found this content?