Blocked Redo Example / Setup / Detection

This example has three scripts that will be used, blockredo_1.sql, blockredo_2.sql, and blockredo_3.sql.

This guideline shows the chronology sequentially on how to set up and view a blocked redo queue scenario.

This example requires an established availability group with a primary and at least one secondary replica. It requires a database replica within the availability group that is configured and replicating to its secondary database replica.

In SQL Server Management Studio, open the three blockredo_x.sql files (1, 2, & 3) into 3 separate query sessions. Make the connections as follows:

Script	Connect To
Blockredo_1.sql	Primary
Blockredo_2.sql	Secondary
Blockredo_3.sql	Secondary

Follow the following steps to set up the blocked redo scenario:

1. **In the session for BLOCKREDO_1.SQL** issue the following commands, change context to a replicated database and issue the following commands to create a table with some records in it:

```
--on primary database (in an availability group)
--create a table
create table blockredo (f1 int)
go
--insert 35K records
insert into blockredo select 1
GO
insert into blockredo select * from blockredo
go 15
select count(*) from blockredo
GO
```

2. **In the session for BLOCKREDO_2.SQL**, change context to the secondary database and issue the following query:

```
select sum(cast(a.f1 as float) + cast(b.f1 as float)) from blockredo a
```

```
cross join blockredo b
cross join blockredo c
group by a.f1
```

The goal of this query is to set up a long-running session that acquires (and holds) a SCH-S lock on the object (table). It is this SCH-S lock that will cause the blocking. On my machine, this query will last at least several minutes – which should give enough time to see the blocked redo scenario.

3. **In the session for BLOCKREDO_3.SQL**, change context to the secondary database and issue the following query:

```
select resource_type, request_mode, request_status
       from sys.dm_tran_locks
       where resource_type = 'OBJECT'
```

It should show results similar to the following:

Results | Messages

	TableName	resource_type	request_mode	request_status
1	blockredo	OBJECT	Sch-S	GRANT
2	blockredo	OBJECT	Sch-S	GRANT
3	blockredo	OBJECT	Sch-S	GRANT
4	blockredo	OBJECT	Sch-S	GRANT
5	blockredo	OBJECT	Sch-S	GRANT

4. **In the session for BLOCKREDO_1.SQL**, issue the next SQL statement to cause a DDL change on the table, BlockRedo. This DDl change will succeed on the primary and cause a SCH-M request on the secondary with the log record is redone on the secondary. On the secondary, the application of this log record will be blocked because of the Sch-S lock held by the query in blockredo_2.sql:

```
--change the datatype (causing DDL operation)
alter table blockredo alter column f1 bigint
```

5. **In the session for BLOCKREDO_3.SQL**, re-execute the query that shows the lock requests:

```
select resource_type, request_mode, request_status
       from sys.dm_tran_locks
       where resource_type = 'OBJECT'
```

You should now see a new lock request of Sch-M, which is in a WAIT status, i.e. blocked:

Results | Messages

	resource_type	request_mode	request_status
1	OBJECT	Sch-S	GRANT
2	OBJECT	Sch-S	GRANT
3	OBJECT	Sch-S	GRANT
4	OBJECT	Sch-S	GRANT
5	OBJECT	Sch-S	GRANT
6	OBJECT	Sch-M	WAIT

6. **In the session for BLOCKREDO_1.SQL**, execute the following to generate some records:

```
update blockredo set f1 = 4
update blockredo set f1 = 5
update blockredo set f1 = 6

SELECT ags.name as AGGroupName,
       ar.replica_server_name as InstanceName,
       hars.role_desc, drs.redo_queue_size,
       CASE drs.is_local WHEN 1 THEN db_name(drs.database_id)
        ELSE NULL END as DBName, drs.database_id,
       ar.availability_mode_desc as SyncMode,
       drs.synchronization_state_desc as SyncState,
       drs.last_hardened_lsn, drs.end_of_log_lsn, drs.last_redone_lsn,
       drs.last_hardened_time, drs.last_redone_time,
       drs.log_send_queue_size, drs.redo_queue_size
       FROM sys.dm_hadr_database_replica_states drs
       LEFT JOIN sys.availability_replicas ar
       ON drs.replica_id = ar.replica_id
       LEFT JOIN sys.availability_groups ags
       ON ar.group_id = ags.group_id
       LEFT JOIN sys.dm_hadr_availability_replica_states hars
       ON ar.group_id = hars.group_id and ar.replica_id =hars.replica_id
       ORDER BY ags.name, group_database_id,
              hars.role_desc, ar.replica_server_name
```

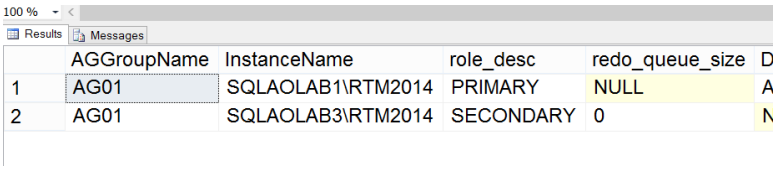
Results | Messages

	AGGroupName	InstanceName	role_desc	redo_queue_size	D
1	AG01	SQLAOLAB1\RTM2014	PRIMARY	NULL	A
2	AG01	SQLAOLAB3\RTM2014	SECONDARY	7764	N

7. **In the session for BLOCKREDO_2.SQL**, cancel the query that has been running all this time, to release the Sch-S locks. This will unblock the redo queue.

8. **In the session for BLOCKREDO_1.SQL**, re-issue the last query to show the redo_queue_size. It should be decreasing or 0 at this point:

```
SELECT ags.name as AGGroupName,
       ar.replica_server_name as InstanceName,
       hars.role_desc, drs.redo_queue_size,
       CASE drs.is_local WHEN 1 THEN db_name(drs.database_id)
        ELSE NULL END as DBName, drs.database_id,
       ar.availability_mode_desc as SyncMode,
       drs.synchronization_state_desc as SyncState,
       drs.last_hardened_lsn, drs.end_of_log_lsn, drs.last_redone_lsn,
       drs.last_hardened_time, drs.last_redone_time,
       drs.log_send_queue_size, drs.redo_queue_size
       FROM sys.dm_hadr_database_replica_states drs
       LEFT JOIN sys.availability_replicas ar
       ON drs.replica_id = ar.replica_id
       LEFT JOIN sys.availability_groups ags
       ON ar.group_id = ags.group_id
       LEFT JOIN sys.dm_hadr_availability_replica_states hars
       ON ar.group_id = hars.group_id and ar.replica_id =hars.replica_id
       ORDER BY ags.name, group_database_id,
             hars.role_desc, ar.replica_server_name
```

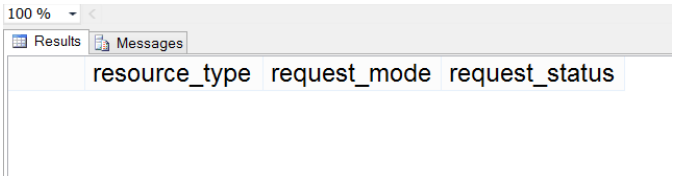


	AGGroupName	InstanceName	role_desc	redo_queue_size	D
1	AG01	SQLAOLAB1\RTM2014	PRIMARY	NULL	A
2	AG01	SQLAOLAB3\RTM2014	SECONDARY	0	N

9. **Finally, in the session for BLOCKREDO_3.SQL**, re-issue the query to show the locks held. The Sch-S and Sch-M locks should be gone.

```
select resource_type, request_mode, request_status
       from sys.dm_tran_locks
       where resource_type = 'OBJECT'
```

You should now see no locks for this table:



	resource_type	request_mode	request_status

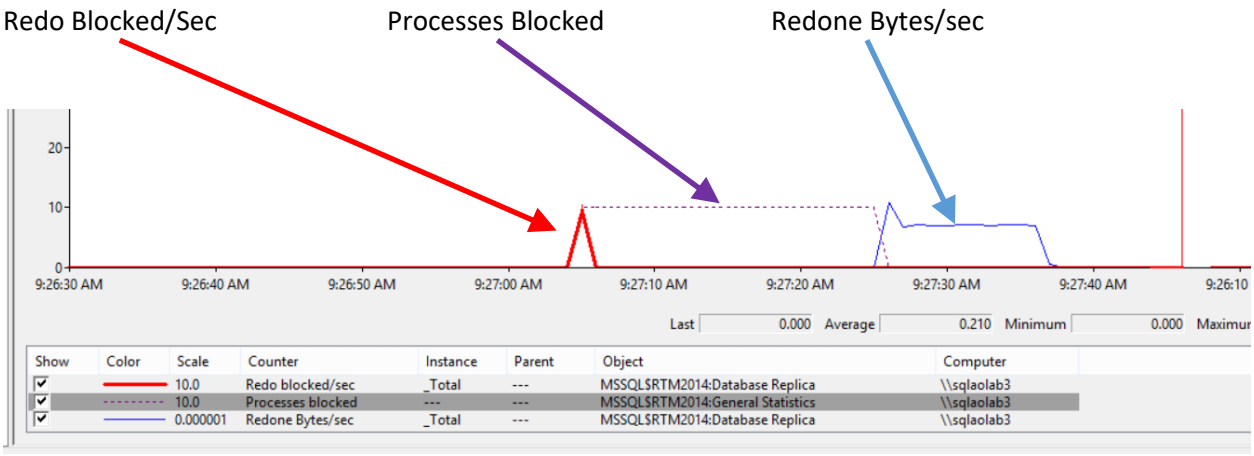
NOTE: In step 4 above, if you had been monitoring the Secondary Replica for the following counters, you could have seen some interesting activity similar to this:

Instance:Database Replica:Redo blocked/sec
Instance:Database Replica:Redone bytes/sec
Instance:General Statistics:Processes Blocked.

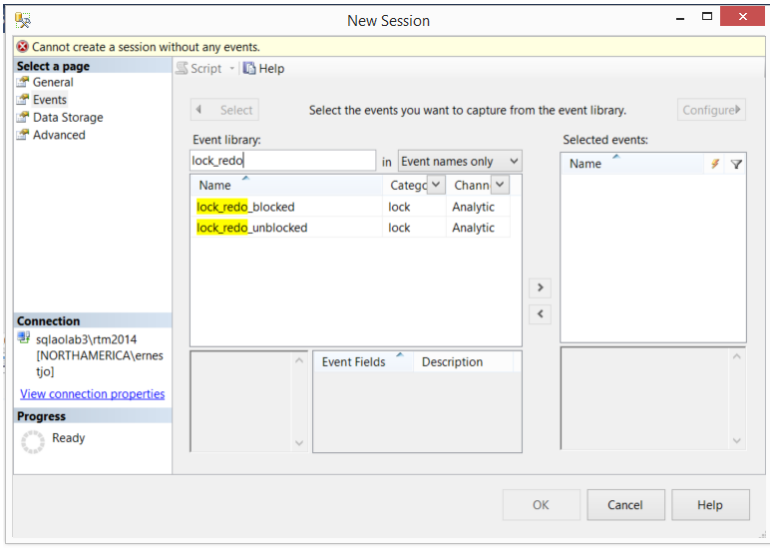
You would see a single spike for the Redo Blocked/Sec counter (bold red below). It does not show a continuous blocking – this counter only registers at the time it happens.

Immediately after the blip for Redo Blocked/sec, you then see the "Processes Blocked" counter (dotted purple line) – which does in fact continually show that the process is blocked.

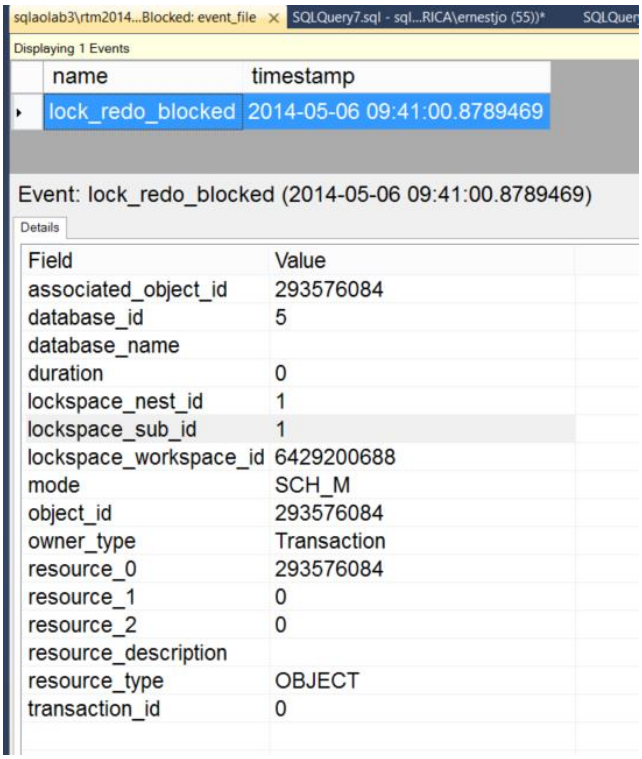
Finally, because our REDO queue had been blocked, when the blocking is finally over and REDO can continue, there could be surge in Redone Bytes/Sec (blue line).



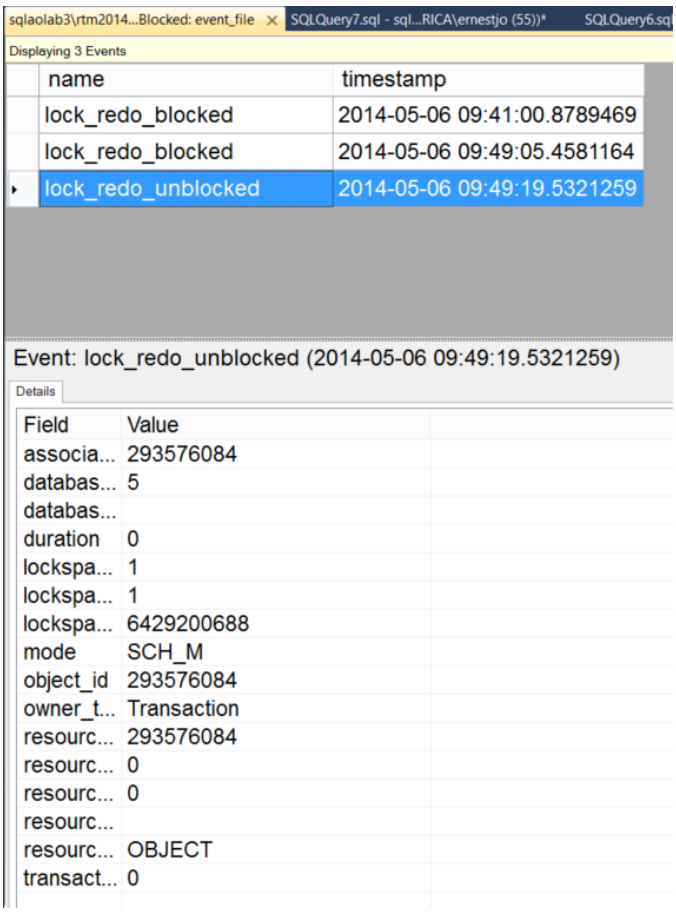
In addition to Perfmon, you can also look for REDO blocking in the AlwaysOn Extended Events. There are two extended events, **lock_redo_blocked** & **lock_redo_unblocked.** In the "Event Library" if you search for "lock_redo" you should find both events.



If these are captured on the secondary replica instance, then it will fire the event if and when REDO blocking occurs. You can get the object ID that was involved in the blocking:



As well as when the blocking is released and REDO continues:



Other references:

Troubleshooting REDO queue build-up (data latency issues) on AlwaysOn Readable Secondary Replicas using the WAIT_INFO Extended Event
https://techcommunity.microsoft.com/t5/sql-server-support-blog/troubleshooting-redo-queue-build-up-data-latency-issues-on/bc-p/3278913#M1177