

## Contents

- Issue
- Replication Agent Architecture overview
- Analysis - Possible Causes for Performance issues in Each S...
  - Log Reader – Reader Thread: reading the transaction log o...
  - Log Reader – Writer Thread: writing to the distribution dat...
  - Distribution Agent – Reader Thread: reading from distribut...
  - Distribution Agent – Writer Thread: writing to Subscriber d...
- Analysis - Tools to assist with performance troubleshooting
  - System Tools
  - Replication Specific Tools
  - Distribution Agent Stats using DMVs
  - SQL Server Management Studio Performance Dashboard
  - Replication Monitor:
  - Replication Tracer Tokens
  - Replication Publication Script
  - Replication Publisher, Distributor, and Subscriber Metadata
  - Log Reader and Distribution Agent Performance Statistics
- Log Reader: Reader-Thread Latency
  - Investigation
    - Scenario: Log Reader appears to hang
    - Scenario: Large batch of replicated commands
    - Scenario: Large number of non-replication transactions
    - Scenario: High number of Virtual Log Files (VLFs)
    - Scenario: Slow Read I/O
  - Mitigation
- Log Reader Writer-Thread Latency
  - Investigation
    - Scenario: Blocking
    - Scenario: High I/O
  - Mitigation
- Distribution Agent Reader Latency
  - Investigation
    - Scenario: Large Batch of Transactions
    - Scenario: High Statement CPU or I/O
  - Mitigation
- Distribution Agent Writer Latency
  - Investigation
    - Scenario: Long Execution Times in Insert/Update/Delete pr...
    - Scenario: User Defined Triggers

- Scenario: 100 many indexes on the Subscriber tables

## Troubleshooting Log Reader and Distribution Agent Performance

- Scenario: Changes not replicated as default parameterized ...

### NOTE

*This information was copied from an article covering on-premise SQL Server. It has been adapted to Managed Instance, but some of the steps and details might not work on MI. Please leave a comment below to suggest any improvement or point out errors. Thank you!*

## Issue

This article provides you with possible causes and steps related to performance issues with replication agents. It discusses various scenarios - if you are unsure which scenario applies to your case, please first review article:

[Performance: Log Reader and Distribution Agent Statistics](#)

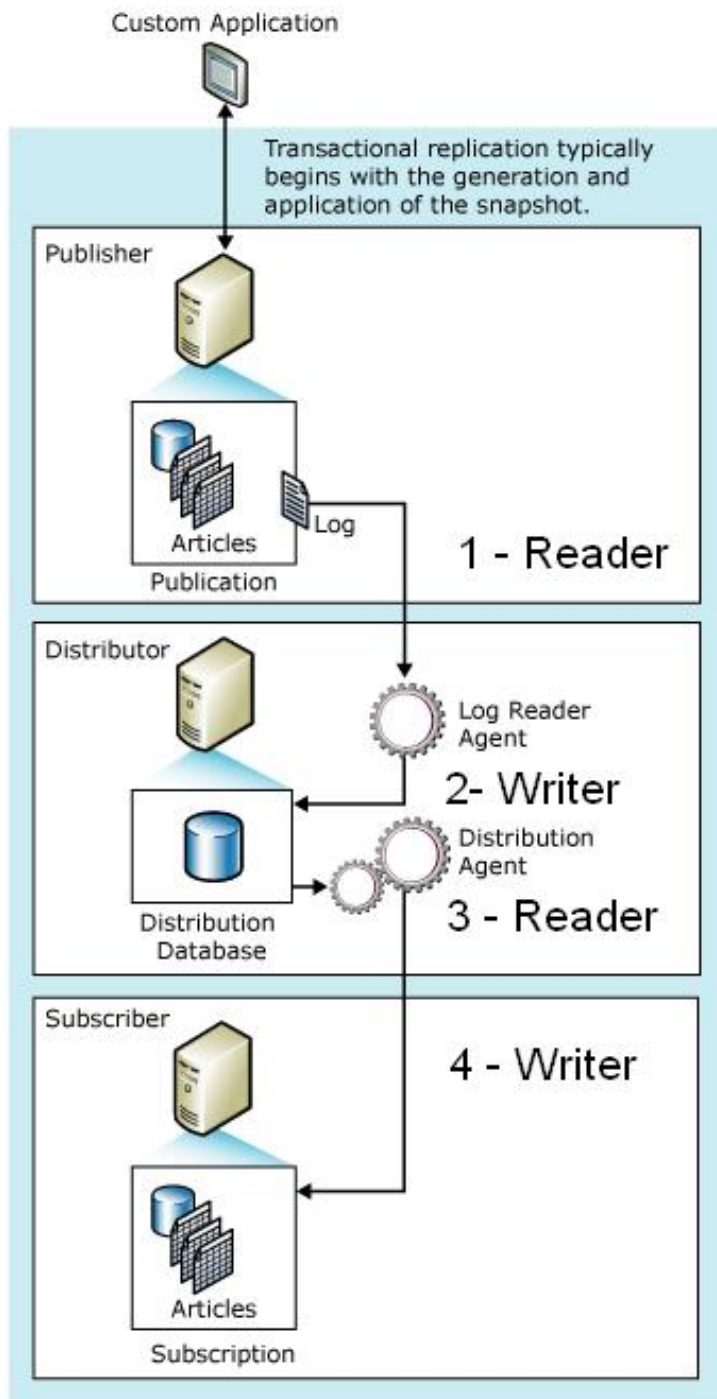
## Replication Agent Architecture overview

You can break down the replication flow into 4 simultaneous data streams. Examine the performance of each data stream to identify where the bottleneck investigation should begin.

The individual data streams are:

1. Log Reader **Reader thread** reads the transaction log of the Publisher database through the stored procedure `sp_replcmds`. It scans the transaction log for transactions marked for replication, skipping the non-replicated transactions. The result is collected in a memory buffer, which is then passed to the writer thread for further processing.
2. Log Reader **Writer thread** receives the replicated transaction in a memory buffer from the reader thread, then writes into the Distribution database using `sp_MSadd_replcmds`.
3. Distribution **Reader thread** executes `sp_MSget_repl_commands` to retrieve pending commands from the Distribution database. The result is collected in a memory buffer, which is then passed to the writer thread for further processing.
4. Distribution **Writer thread** receives the replicated transaction in a memory buffer from the reader thread, then writes them to the Subscriber database through parameterized stored procedures prefixed with `sp_MSupd...` - `sp_MSins...` - `sp_MSdel...` to apply individual row changes to each article at the subscriber.
5. Log Reader and Distribution Agents also write status information to a history thread into tables `MSlogreader_history` and `MSdistribution_history` in the distribution database.

And picture says more than words:



For more details of what specific steps the agents take, see [Transactional Replication Process Flow]

## Analysis - Possible Causes for Performance issues in Each Scenario

### Log Reader – Reader Thread: reading the transaction log of the Publisher database

Symptom: Log Reader appears to hang or is progressing slowly. Message "hasn't logged a message for the past xxx minutes"

Possible causes:

- SQL/SOS Scheduler issues - non-yielding scheduler, deadlocked scheduler, etc.

- Large batch of replicated transaction and commands in the Publisher database (it will take a long time to process). Possibly bulk-loaded, inserted, updated, or deleted a lot of data
- Large number of non-replication transactions and commands. The Log Reader is reading sequentially from the beginning of the transaction log, thus takes a long time to read the transaction log to find the next replicated transactions.
- High number of Virtual Log Files (VLFs) - *not sure if this still applies to MI*
- LOGLFM spinlock wait - *not sure if this still applies to MI*

Typical stored procedure activity:

- sp\_replcmds/xp\_replcmds
- sp\_repldone
- sp\_MSadd\_logreader\_history
- sp\_MSPub\_adjust\_identity

## Log Reader – Writer Thread: writing to the distribution database

Possible causes:

- Blocking against the distribution database tables, mainly: `MSrepl_transactions` , `MSrepl_commands` , `MSlogreader_history` , `MSdistribution_history` . Often caused by Distribution Cleanup job blocking the replication agents.
- High I/O, insufficient IOPS at the Distributor causing slow DML in the distribution database/tables
- Writelog waits in the Distribution database
- Network I/O, might apply if Publisher and Distributor are on different Managed Instances (Remote Distributor configuration)

Typical stored procedure activity:

- sp\_MSadd\_replcmds
- sp\_MSget\_last\_transaction

## Distribution Agent – Reader Thread: reading from distribution database

Symptom: `sp_MS_get_repl_commands` taking a long time to read replicated transactions.

Possible causes:

- Blocking against the distribution database tables, mainly: `MSrepl_transactions` , `MSrepl_commands` , `MSlogreader_history` , `MSdistribution_history` . Often caused by Distribution Cleanup job blocking the replication agents.
- It is trying to read a large transaction with a lot of commands
- The execution plan for reading from `MSrepl_transactions` and/or `MSrepl_commands` is scanning the tables:
  - the tables are very large (is the cleanup task working and able to catch up?)
  - outdated statistics, fragmentation
- Memory pressure at the Distributor, needs to read data from storage, thus affected by insufficient IOPS at the Distributor

Typical stored procedure activity:

- sp\_MSget\_repl\_commands
- sp\_MShelp\_distribution\_agentid
- sp\_MSget\_subscription\_guid

## Distribution Agent – Writer Thread: writing to Subscriber database

Symptoms:

- slow write performance at the Subscriber database, unable to catch up with the incoming workload
- long execution times or high I/O

Possible causes:

- Bad execution plan of the Insert/Update/Delete stored procedures
- Too many custom indexes on the Subscriber tables (e.g. added to help reporting performance)
- User Defined Triggers on the Subscriber tables
- Network latency, if the Subscriber is on another MI or different environment
- Distribution Agent taking a long time to apply the initial snapshot (consider using Subscription Streams)

Typical stored procedure activity:

- sp\_MSins%
- sp\_MSupd%
- sp\_MSdel%

## Analysis - Tools to assist with performance troubleshooting

### System Tools

- Distribution Agent Stats using DMVs
- SQL Server Management Studio Performance Dashboard

### Replication Specific Tools

- Replication Monitor
- Replication Tracer Tokens
- Replication Publication Script
- Replication Publisher, Distributor, and Subscriber Metadata
- Log Reader and Distribution Agent Performance Statistics

### Distribution Agent Stats using DMVs

If you have identified one or more queries that are involved in the performance issue, you can pull the query plan from the server using system DMVs to retrieve cached query plans. For example, to pull all activity against the Distribution database, execute:

([http://mswikis/SQLTroubleshooting/Pages/Sample Script to Collect dm\\_exec\\_query\\_stats.aspx](http://mswikis/SQLTroubleshooting/Pages/Sample Script to Collect dm_exec_query_stats.aspx) )

```
--dm_exec_query_stats for Log Reader Writer-Thread sp_MSadd_replcmds
-- by top total_worker_time
SELECT TOP 25
st.text, qp.query_plan,
(qs.total_logical_reads/qs.execution_count) as avg_logical_reads,
(qs.total_logical_writes/qs.execution_count) as avg_logical_writes,
(qs.total_physical_reads/qs.execution_count) as avg_phys_reads,
qs.*
FROM sys.dm_exec_query_stats as qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
WHERE st.text like '%distribution%'
ORDER BY qs.total_worker_time DESC
Go
```

Other possible Replication SPs search criteria:

```
WHERE st.text like 'CREATE PROCEDURE sys.sp_MSget_repl_commands%'
WHERE st.text like 'CREATE PROCEDURE sp_MShistory_cleanup' -- Distribution cleanup
WHERE st.text like 'CREATE PROCEDURE sp_MSdistribution_delete%' -- Distribution cleanup
WHERE st.text like '%sp_repldone%' -- Log Reader Agent on Publisher database
WHERE st.text like '%sp_MSadd_replcmds%' -- Log Reader Agent on Distribution database
WHERE st.text like '%distribution%' -- Distribution database Reads/Writes
WHERE st.text like '%sp_MSupd%' -- Distribution Agent Updates on Subscriber
WHERE st.text like '%sp_MSins%' -- Distribution Agent Inserts on Subscriber
WHERE st.text like '%sp_MSdel%' -- Distribution Agent Deletes on Subscriber
```

## SQL Server Management Studio Performance Dashboard

The SSMS Performance Dashboard Reports are Reporting Services reports designed to provide an overview about the current performance metrics, and help find blocking, disk bottlenecks, and statements causing high IO and high CPU.

You can reach these reports in SSMS by right-clicking on the Managed Instance server - Reports - Standard Reports, then selecting either the Performance Dashboard or one of the detail reports.

The reports allow a database administrator to quickly identify whether there is a current bottleneck on their system, and if a bottleneck is present, capture additional diagnostic data that may be necessary to resolve the problem. For example, if the system is experiencing waits for CPU or Disk IO, the dashboard allows the admin to quickly see which sessions are performing the most CPU or IO, what query is running on each session and the query plan for each statement.

Common performance problems that the dashboard reports may help to resolve include:

- CPU bottlenecks (and what queries are consuming the most CPU)
- IO bottlenecks (and what queries are performing the most IO).
- Index recommendations generated by the query optimizer (missing indexes)
- Blocking
- Latch contention

The information captured in the reports is retrieved from SQL Server's dynamic management views. There is no additional tracing or data capture required, which means the information is always available and this is a very

inexpensive means of monitoring your server.

To help identify Replication components, look for the Log Reader or Distribution Agent job names as the Program\_Name connecting to SQL Server. To see a list of the Replication Jobs, run the following query:

```
SELECT name, description, enabled from msdb..sysjobs
WHERE category_id > 10 and category_id < 20
```

Replication Monitor:

This tool provides overview of the Replication Agent status and is a good first-stop for examining overall replication topology and Agent health. Note some of these features query the same system tables as the Replication Agent. If the Agents are having performance problems, expect this tool to have issues as well.

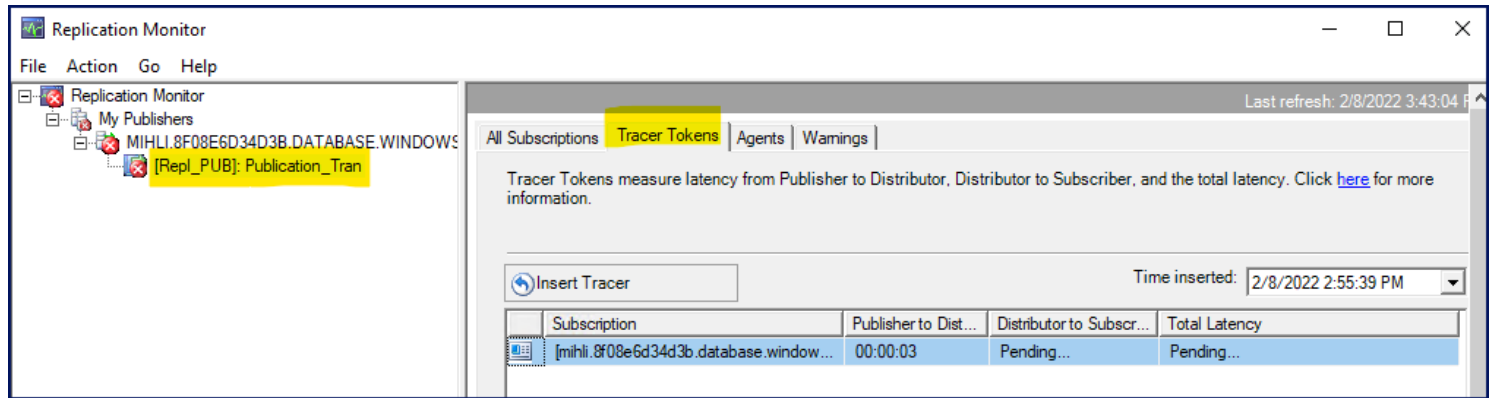
You can reach Replication Monitor through SSMS: connect to the Publisher/Distributor - expand the server node - right-click on the Replication node or any of its subnodes.

Feature Summary:

- Current status of Log Reader and Distribution Agents
- History of Agent executions
- Ability to set Agent Properties and Profiles for unique configurations
- Start and Stop Agent
- Insert Tracer Tokens to monitor replication latency
- Display number of commands in the distribution database waiting to be replicated
- Display estimated time to apply pending commands

Replication Tracer Tokens

To gain a quick high-level overview of Replication performance, use the Tracer Token feature in Replication Monitor. The features provides “Publisher to Distributor” and “Distributor to Subscriber” latency values. To use this feature, launch Replication Monitor. In the My Publishers list, drill in and click on the publication you wish to test. In the right pane, choose the Tracer Token tab and click the “Insert Tracer Token” button. There will be a list of subscriptions below. Monitor the “Publisher to Distributor” and “Distributor to Subscriber” values.



This method is only helpful in situations where data is still flowing somewhat real-time. If the topology is behind 3 days, then you will never see the tracer token make it to the subscriber in time. Tracer Tokens show if the replication agents are keeping up or getting behind, and at what step the latency is introduced:

- If you are seeing a long time for “Publisher to Distributor” to populate, focus on troubleshooting the Log Reader performance.
- If Log Reader performance is fast while the “Distributor to Subscriber” latencies are high, you should look into possible Distribution Agent bottlenecks.
- It is also possible that both are slow, from either the same or different causes, and that bad Log Reader performance may hide a bigger issue at the Distribution Agent.

## Replication Publication Script

SSMS has a feature to generate a script file with the SQL statements needed to rebuild the replication environment. These script contain current Publisher, Distributor, and Subscriber settings and contain information such as the number of publications, listing of articles per publication, horizontal or vertical filtering, and location of the Snapshot files.

You should collected such a script when you cannot resolve the issue through a performance analysis and suspect that the issue might be related to the design and size of the publication or the overall setup. For example, the replication script would show you the article options about indexes, constraints, or triggers, and show you any custom scripts or procedures that are part of the publication configuration.

To generate the Replication script:

- Start SQL Server Management Studio and connect to each server participating in the replication topology
- Right-click the Replication folder, then “Generate Scripts”
- The default settings to script Distributor and all Publications provides best overall picture
- Click “Script to file”.

## Replication Publisher, Distributor, and Subscriber Metadata

A valuable resource while investigating replication performance are the replication metadata tables. The most important of these are the Replication Agent history and MSrepl\_errors tables stored in the Distribution database. For nearly all situation, use the script below to collect the replication Distribution metadata tables into a SQL database which can then be queried. If time permits, also collect the Publisher and Subscriber metadata.

```
USE <distribution database>
SELECT * FROM dbo.MSpublications
SELECT * FROM dbo.MSarticles
SELECT * FROM dbo.MSlogreader_agents
SELECT * FROM dbo.MSlogreader_history
SELECT * FROM dbo.MSdistribution_agents
SELECT * FROM dbo.MSdistribution_history
SELECT * FROM dbo.MSsubscriptions
SELECT * FROM dbo.MStracer_tokens
SELECT * FROM dbo.MStracer_history
SELECT * FROM dbo.MSrepl_errors
```

By default, the history cleanup process is run every 10 minutes and history records older than 48 hours are deleted. If troubleshooting takes longer than just a few hours, you should

- either stop and disable the history cleanup job "Agent history clean up: Distribution" (do not stop the regular cleanup "Distribution clean up: Distribution!")
- or increase the History retention periods through the Distributor properties:



1. Using SQL Server Management Studio, right-click the Replication Folder
2. Select Distributor Properties
3. Under the list of Distribution Database, click the "... " button for each distribution database. Note, there may be only 1 database listed
4. Increase the History Retention to enough days to cover potential problems recoded by the Replication Agents.

## Log Reader and Distribution Agent Performance Statistics

Performance statistics are recorded for each agent in the history tables for the Log Reader Agent and for the Distribution Agent every five minutes. The following is a sample performance output from the history table for the Log Reader Agent:

```
<stats state="1" work="9" idle="295">  
<reader fetch="8" wait="0"/>  
<writer write="9" wait="0"/>  
<sinclaststats elapsedtime="304" work="9" cmds="52596" cmdspers="5753.000000">  
<reader fetch="8" wait="0"/>  
<writer write="9" wait="0"/>  
</sinclaststats>  
</stats>
```

More info on how to capture and interpret these stats can be found at: [Performance: Log Reader and Distribution Agent Statistics](#)

## Log Reader: Reader-Thread Latency

### Investigation

The Log Reader reader-thread is reading the transaction log via the stored procedure `sp_replcmds`, which is a wrapper for the extended stored procedure `xp_replcmds`. It scans the transaction log for transactions marked for replication, skipping non-replicated transactions. High latency in the reader thread most often occurs when processing large batches of transactions, when either the number of replicated commands is very high (e.g. million of replicated changes in one transaction), or when it has to read through a lot of non-replicated commands to find the next replicated command.

Use troubleshooting steps below to determine if the Log Reader Agent is:

- stuck or hung,
- slow in reading the transaction log
- slow writing to the distribution database
- blocked writing updates to the distribution history table

### Scenario: Log Reader appears to hang

The progress of the Log Reader is reflected in the output of DBCC OPENTRAN, which keeps track of the oldest distributed Log Sequence Number (LSN) and the oldest non-distributed LSN. When the Log Reader is moving transactions, the DBCC OPENTRAN values will change very quickly. If the oldest distributed LSN appears to change very slowly or not at all, it might indicate a performance issue with the Log Reader .

Run DBCC OPENTRAN in the Publisher database:

```
DBCC OPENTRAN
-- sample output:
Transaction information for database 'TranPub'.
Replicated Transaction Information:
Oldest distributed LSN : (2074:387:4)
Oldest non-distributed LSN : (2074:487:1)
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

When all replicated transactions have been moved from the log to the distribution database, the counter shows zeros:

```
Oldest non-distributed LSN : (0:0:0)
```

As the Log Reader moves through the transaction log, it will also be updating first\_begin\_lsn:

```
-- run in Publisher database:
select first_begin_lsn, * from sys.dm_repl_tranhash
```

Check `sys.dm_exec_sessions` if the Log Reader is progressing. From the session stats, make note of the `login_time`, `status`, `cpu_time`, `memory_usage`, `reads`, `writes`, `logical_reads`, etc. to confirm that these values are changing, which indicates it is indeed running and not stalled.

```
SELECT * FROM sys.dm_exec_sessions
WHERE program_name LIKE 'Repl-LogReader%'
```

Although it is not always desirable to interrupt the Log Reader Agent, you might consider configuring the agent with an output file. The output file will record additional performance statistics that allows you to see which steps are taking longest. The disadvantage is that you need to restart the agent, and if it was busy processing commands, it will have to redo the work after the restart. Also, if you provide an invalid file path, the agent will fail to start, thus causing additional delays.

Stop the Log Reader Agent, add the “-Output” parameter with a valid Azure File Storage path, then restart the Agent. See article [Add verbose logging to Replication Agents in MI](#) for further details.

The output file breaks down the time for both the Reader (Fetch time) and the Writer (Write time) streams. In the sample output below, the Reader thread took 55 minutes while the Writer thread needed only 3 minutes to process the transaction, indicating Log Reader reader thread is the bottleneck.

```
***** STATISTICS SINCE AGENT STARTED *****
Execution time (ms): 3401930
Work time (ms): 3401470
Distribute Repl Cmds Time(ms): 3354520
Fetch time(ms): 3343920 <-- read time 55 minutes
Repldone time(ms): 5520
Write time(ms): 140070 <-- write time 3 minutes
Num Trans: 8 Num Trans/Sec: 0.002352
Num Cmds: 477219 Num Cmds/Sec: 140.297871
```

In this example, the Log Reader was not hung, but took 55 minutes to read through the transaction log and return set of commands to be replicated.

Further questions to check:

- How large is the transaction Log for this published database?
- Has it grown to an unexpected size causing longer read times by the Log Reader thread?

If the size of the Publisher database transaction log is significantly larger than expected, the Log Reader may require additional time to scan the log. Check the log size of the Publisher database and compare it with other databases on the same server instance:

```
DBCC SQLPERF(LOGSPACE)
```

The Log Reader records its status messages through the Log Reader History Writer-thread, which then writes these messages to the `MSrepl_errors` and `MSlogreader_history` tables in the distribution database. If the History Writer-Thread is blocked, then the primary Reader and Writer threads must wait until the messages buffer has been cleared. This may appear as short-term Log Reader latency.

The History-Writer blocking may not be severe enough to show in the Blocker output. However, it should show in cumulative wait stats.

Examine system wait status to see if there is long wait on the Distribution database.

### Scenario: Large batch of replicated commands

The reader thread of the Log Reader executes `sp_replcmds` to pull commands from the transaction log. The execution time for this stored procedure approximates the startup time for the Log Reader to replicate this same transaction. Scanning a large transaction log for a single transaction containing several million row modifications may take 20 minutes and more to complete.

If you suspect that `sp_replcmds` is slow to return results, you may execute it yourself from an SSMS query window to check its performance. This may help answering the following questions:

- How long does it take to return the next replicated transaction?
- How many commands are returned in this transaction?
- How long does it take to return the default batch size of 500 transactions?
- Is this set of commands part of the normal database operations or unexpected?

Steps to run `sp_replcmds` and the correlated `sp_replshowcmds` :

- Stop the Log Reader Agent job (note that this cancels any of its pending read or write operations)
- Connect with a query window to the Publisher database
- Execute the following set of commands and make sure to include executing `sp_replflush`
- Start the regular Log Reader Agent job again .

```

use <publisher database>
GO

-- return all commands for the next pending transaction
exec sp_replcmds @maxtrans = 1
GO
-- return all commands for the next 500 transactions
exec sp_replcmds @maxtrans = 500
GO
-- executing sp_replcmds marks the connection as Log Reader; unmark and reset this now by calling:
exec sp_replflush
GO

-- return all commands for the next pending transaction with the command text
-- commands type = 1073741871 are pending Tracer Tokens
sp_replshowcmds @maxtrans = 1
GO

-- return all commands for the next 500 transactions with command text
sp_replshowcmds @maxtrans = 500
GO

-- executing sp_replcmds marks the connection as Log Reader; unmark and reset this now by calling:
exec sp_replflush
GO

```

Another technique to see a summary of pending transactions/commands is to query the `sys.dm_repl_traninfo` system view. It shows you data on replicated transactions that were last read by `sp_replcmds` or `sp_replshowcmds`, including the "cmds\_in\_tran". A high value may indicate a large transaction is being replicated.

```

SELECT dbid, begin_lsn, commit_lsn, rows, cmds_in_tran
FROM sys.dm_repl_traninfo

```

dbid	begin_lsn	commit_lsn	rows	cmds_in_tran
10	0000081A:0000013A:0001	0000081A:0000013A:0004	1	1
10	0000081A:00000140:0001	0000081A:00000140:0004	1	1
10	0000081A:00021385:0001	0000081E:00004EA2:0009	6874	6874

The following commands can be used to determine the number of non-replicated vs. replicated transactions. A transaction log with a high percentage of non-replicated transaction will cause latency as the Log Reader is scanning over transactions it has to ignore. These can be database maintenance transactions, such as an online reindex, or data modifications to tables that are not being replicated. A 25 GB transaction log may take up to 15 minutes or more to scan, depending on the I/O configuration of the Managed Instance.

```





use <publisher database>
GO

-- total number of records in the log
SELECT count(*) FROM ::fn_dblog(NULL, NULL)
GO

-- records marked for REPLICATION
SELECT count(*) FROM ::fn_dblog(NULL, NULL) WHERE Description='REPLICATE'
GO

```

## References:

- [sp\\_replcmds](#) 
- [sp\\_replshowcmds](#) 
- [sp\\_replflush](#) 
- [sys.dm\\_repl\\_traninfo](#) 

## Scenario: Large number of non-replication transactions

This can be more challenging to identify. The best indication is the Log Reader history and its progress messages. It also helps to have a good knowledge of which articles are being replicated and what batch changes have been recently performed.

The output below from the Log Reader history in `distribution..MSlogreader_history` clearly shows how many rows are being read while only a few are being replicated:

```
2008-11-11 23:41:35.527 The Log Reader Agent is scanning the transaction log for commands to be replicated.
Approximately 500000 log records have been scanned in pass # 3, 142 of which were marked for replication, elap
```



## Scenario: High number of Virtual Log Files (VLFs)

A large number of Virtual Log Files (VLFs) can contribute to long running read times for the Log Reader. To determine the number of VLFs, execute the following command and note the number of segments (rows) returned. Segment counts in 100K+ may be contributing to Log Reader Reader-Thread performance problems.

```
USE <publisher database>
GO
DBCC LOGININFO
GO
```

## Scenario: Slow Read I/O

A slow or overloaded database storage may be a contribution to the long read times. Check the performance metrics on the ASC Troubleshooter for indications that the Publisher database is reaching I/O capacity.

## Mitigation

- If the Reader Thread latency is caused by a large number of pending replicated commands, then waiting for the Log Reader to catch up may be the best short-term solution.
- If you suspect that large transactions are pending on the transaction log, consider running the Log Reader Agent with `ReadBatchSize = 1` until the large transactions have been processed. Scanning for the default number of 500 replicated transactions might take longer than the configured timeout but might work if it needs to read just one transaction. Monitor the Agent -OUTPUT file, the `MSlogreader_history` table, or the `DBCC OPENTRAN` output for the `sp_repldone` to mark the transaction as replicated.
- Large number of non-replication transactions: This is unavoidable when there are large numbers of transactions and commands for non-replicating tables. If the problem is caused by index maintenance log

records, consider performing offline reindexing. Another option may be stopping and starting the online reindexing to create smaller batch of index update transactions.

- High number of VLFs: Shrink the transaction log to move extra VLFs. Set the database "growth" to value which allows for growing without creating high number of VLFs
- Slow Read I/O: ask the customer to scale to Business Critical for reducing I/O latency and getting SSD I/O throughput.
- Depending on estimated time to read pending transactions and transfer them to the subscribers, it may be faster to mark all transactions as "replicated", then reinitialize the subscribers with a new snapshot or via backup/restore. This is step should only be taken if the time to generated a new snapshot and to deliver it to the Subscriber is faster than waiting for individual pending commands to be replicated.

Note though that this affects all publications and all Subscribers to these publications - all Subscribers need to be reinitialized after this step!

Steps to mark all transactions as replicated:

```
EXEC sp_repldone @xactid = NULL, @xact_segno = NULL, @numtrans = 0, @time = 0, @reset = 1
```

Reference: [sp\\_repldone](#) 

## Log Reader Writer-Thread Latency

### Investigation

Check `distribution..MSlogreader_history` for the Log Reader performance statistics. If you see `<stats state="2"` events, it indicates that the agent is taking a long time to write changes to the destination.

Please see section "Log Reader and Distribution Agent Performance Statistics" above to get more details.

Check the performance metrics on the SSMS Performance Dashboard reports for indications that

`sp_MSadd_replcmds` (the stored procedure of the writer thread) is involved in any issue like long execution times, high CPU, or blocking.

Another indicator would be the performance statistics on the agent's output file. See article [Add verbose logging to Replication Agents in MI](#) for further details

Get an overview about DML activity per article - is this data volume expected for these tables?

```
--Get a list of articles ID for the Publisher database:
use <publisher database>
GO
exec sp_helptarticle 'name of publication'
GO

select publisher_db, article_id, count(article_id) as 'RowCount'
FROM distribution.dbo.MSrepl_commands with (nolock)
JOIN distribution.dbo.MSpublisher_databases
ON publisher_database_id = id
group by publisher_db,article_id
```

publisher_db	article_id	RowCount
TranPub	0	1
TranPub	1	2734567
TranPub	2	19
TranPub	3	15

(4 row(s) affected)

This example clearly shows 1 table is accounting for 99% of all traffic in the Distribution database. If this is unexpected, then check with the customer what was done to create this workload.

### Scenario: Blocking

Blocking may be caused by another replication agent such as a the Distribution Agents, the Distribution Cleanup job, or the Distribution History cleanup job. Investigate blocking by using the usual SQL tools, e.g. the SSMS Activity Monitor, SSMS Performance Dashboard reports, sysprocesses, or sp\_who2.

Check `sys.dm_exec_sessions` if the Log Reader is progressing. From the session stats, make note of the login\_time, status, cpu\_time, memory\_usage, reads, writes, logical\_reads, etc. to confirm that these values are changing, which indicates it is indeed running and not stalled:

```
SELECT * FROM sys.dm_exec_sessions
WHERE program_name LIKE 'Repl-LogReader%'
```

An excessive number of rows in the distribution metadata tables will contribute to longer execution times and will increase the opportunity for blocking. Check the size of the main Distribution database tables; look for high row counts (> 1 million), which might point to cleanup not running properly, or might indicate large pending transactions.

```
-- Distribution Agent cleanup running?
SELECT COUNT(*) FROM distribution.dbo.MSrepl_commands WITH (INDEX=ucMSrepl_commands)
SELECT COUNT(*) FROM distribution.dbo.MSrepl_transactions WITH (INDEX=ucMSrepl_transactions)
-- Distribution History cleanup running?
SELECT COUNT(*) FROM distribution.dbo.MSlogreader_history WITH (INDEX=ucMSlogreader_history)
SELECT COUNT(*) FROM distribution.dbo.MSdistribution_history WITH (INDEX=ucMSdistribution_history)
SELECT COUNT(*) FROM distribution.dbo.MSrepl_errors WITH (INDEX=ucMSrepl_errors)
```

Or use a lower-impact check through sysindexes:

```
SELECT name, rowcnt, STATS_DATE (id, indid) as 'Last Update Stats'
FROM distribution.dbo.sysindexes
WHERE name IN('ucMSrepl_transactions', 'ucMSrepl_commands')
```

## Scenario: High I/O

A slow or overloaded database storage may be contributing to the long write times. Check the performance metrics on the ASC Troubleshooter for indications that the Distribution database is reaching I/O capacity.

Also check ASC Troubleshooter for queries that might have changed execution plans. High I/O and/or high CPU may be caused by out-of-date table statistics and thus bad query execution plans.

Check the system DMVs for details about duration and CPU of the Log Reader Writer-Thread. Low Duration and CPU but high Logical Reads may indicate a poor query execution plan caused by out of data table statistics. The command below will retrieve the query plan along with the execution statistics.

```
-- dm_exec_query_stats for Log Reader Writer-Thread sp_MSadd_replcmds
-- by top total_worker_time
SELECT TOP 25
st.text, qp.query_plan,
(qs.total_logical_reads/qs.execution_count) as avg_logical_reads,
(qs.total_logical_writes/qs.execution_count) as avg_logical_writes,
(qs.total_physical_reads/qs.execution_count) as avg_phys_reads,
qs.*
FROM sys.dm_exec_query_stats as qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
WHERE st.text like 'CREATE PROCEDURE sp_MSadd_replcmds%'
ORDER BY qs.total_worker_time DESC
```

## Mitigation

Blocking:

- If the “head blocker” is the Distribution cleanup, consider stopping this Agent, allowing data to get replicated, then restart cleanup during off-peak hours. Blocking may indicate I/O taking longer than expected to complete. See “High I/O” for additional troubleshooting steps.
- Verify Distribution Agent and the Distribution Cleanup jobs are running.

High I/O:

- Check when the statistics were last updated on replication system tables; run Update Statistics if it is too old:

```
DBCC SHOW_STATISTICS(MSrepl_transactions,ucMSrepl_transactions)
DBCC SHOW_STATISTICS(MSrepl_commands,ucMSrepl_commands)

UPDATE STATISTICS MSrepl_transactions WITH FULLSCAN
UPDATE STATISTICS MSrepl_commands WITH FULLSCAN
```

- Retrieve Distribution database settings and check if auto-update statistics is enabled:

```
-- Look for IsAutoUpdateStatistics:
exec sp_helpdb distribution
-- or:
select DATABASEPROPERTYEX('distribution','IsAutoUpdateStatistics') -- (returns 1 if enabled)
```



## Distribution Agent Reader Latency

The Distribution Agent Reader-Thread is responsible for querying the Distribution database for the list of transactions that need to be applied at the Subscriber.

The agent first queries the Subscriber metadata table to find the last successfully-replicated transaction. The Reader-Thread then executes `sp_MSget_repl_commands` to begin reading rows from the Distribution database. It fills the transactions and commands into a memory buffer and passes that over to the writer thread, which then begins writing these rows to the Subscriber.

### Investigation

Check `distribution..MSdistribution_history` for the Distribution Agent performance statistics. You may see output like the following and interpret the `wait` times to see if either reader or writer thread is causing a delay:

```
<stats state="1" work="14798" idle="2035">
  <reader fetch="14798" wait="193"/>
  <writer write="12373" wait="9888"/>
  <sinclaststats elapsedtime="424" work="415" cmds="296900" cmdsperssec="713.000000">
    <reader fetch="415" wait="7"/>
    <writer write="377" wait="212"/>
  </sinclaststats>
</stats>
```

In this example, the `reader fetch wait` is much lower than the `writer write wait`. It indicates that the reader thread is busy getting its task done, whereas the writer thread is idle waiting for the reader to provide data. Please see section "Log Reader and Distribution Agent Performance Statistics" above to get more details.

Check the performance metrics on the SSMS Performance Dashboard reports for indications that `sp_MSget_repl_commands` (the stored procedure of the reader thread) is involved in any issue like long execution times, high CPU, or blocking.

Although it is not always desirable to interrupt the Distribution Agent, you might consider configuring the agent with an output file. The output file will record additional performance statistics that allows you to see which steps are taking longest. The disadvantage is that you need to restart the agent, and if it was busy processing commands, it will have to redo the work after the restart. Also, if you provide an invalid file path, the agent will fail to start, thus causing additional delays.

Stop the Distribution Agent, add the `"-Output"` parameter with a valid Azure File Storage path, then restart the Agent. See article [Add verbose logging to Replication Agents in MI](#) for further details.

The output file breaks down the time for both the Reader (Fetch time) and the Writer (Write time) streams.

```

***** STATISTICS SINCE AGENT STARTED *****
03-22-2009 09:55:19
Total Run Time (ms) : 59189906 Total Work Time : 492741
Total Num Trans : 5 Num Trans/Sec : 0.01
Total Num Cmds : 5 Num Cmds/Sec : 0.01
Total Idle Time : 58660470
Writer Thread Stats
Total Number of Retries : 0
Time Spent on Exec : 0
Time Spent on Commits (ms): 16 Commits/Sec : 0.05
Time to Apply Cmds (ms) : 0 Cmds/Sec : 0.00 <----- Writer Thread
Time Cmd Queue Empty (ms) : 528717 Empty Q Waits > 10ms: 5
Total Time Request Blk(ms): 59189187
P2P Work Time (ms) : 0 P2P Cmds Skipped : 0
Reader Thread Stats
Calls to Retrieve Cmds : 11738
Time to Retrieve Cmds (ms): 492741 Cmds/Sec : 0.01 <----- Reader Thread
Time Cmd Queue Full (ms) : 0 Full Q Waits > 10ms : 0
*****

```

This -OUTPUT taken from the Distribution Agent log shows 100% of the duration was spent reading rows from the Distribution database before the Agent was stopped.

The Distribution Agent log also records time when events completed. In the example below, 4 minutes elapsed between time the Distribution Agent queries the Subscriber for the starting transaction to replicate and time it retrieved a batch of transactions from the Distribution database. The latency is caused by Reader Threads are waiting for CMDs to be returned from the Distribution database.

```

--Read for Subscription metadata completes
2009-05-16 01:26:49.229 OLE DB Distributor 'SQL380': {call sp_MSget_subscription_guid(11)}
--Four minutes later read of replication cmds completes.
2009-05-16 01:29:31.401 sp_MSget_repl_commands timestamp value is: 0x0x000081a0000004500040000000
--Reader threads are waiting for CMDs to be returned from the Distribution DB.

```

To test the READER execute times, run `sp_MSget_repl_commands` from a Query window and look for blocking, table scans, timeouts, etc.:

1. From the -OUTPUT get the LSN for the long running `sp_MSget_repl_commands`  
2009-05-16 01:29:31.401 sp\_MSget\_repl\_commands timestamp value is: 0x0x000081a0000004500040000000
2. Get the Agent\_ID from the same log
3. Execute `sp_MSget_repl_commands` query against Distribution DB  
exec sp\_MSget\_repl\_commands @agent\_id=11, @last\_xact\_seqno=0x000081a0000004500040000000

You can also retrieve the last successful transaction from the subscriber's database using query below. The `transaction_timestamp` value contains the Log Sequence Number (LSN) used for the `sp_MSget_repl_commands` stored procedure.

```

USE <subscriber db>
-- Retrieve last successful Transaction
-- multiple rows per publication indicate parallel distribution streams
SELECT publisher,publication, distribution_agent,transaction_timestamp
FROM dbo.MSreplication_subscriptions

```

## Scenario: Large Batch of Transactions

Latency problems are often caused when series of transactions are trying to move a large batch of commands to the Subscribers. The queries below show overall row counts and index statistics for commands stored in the Distribution database.

```
-- Look for high row counts (> 1 million) indicate cleanup not running
-- or large pending transactions.
SELECT name, rowcnt, STATS_DATE (id, indid) as 'Last Update Stats'
FROM distribution.dbo.sysindexes
WHERE name IN('ucMSrepl_transactions', 'ucMSrepl_commands')
```

Are the row counts expected or do they now contain millions of rows? High row counts (> 1 million) may indicate a large transaction is being processed; or that the Distribution cleanup procedure is not running or keeping up with the workload.

When performance troubleshooting latency a review of pending commands by day by # commands may uncover helpful patterns. A breakdown of the commands being stored in the Distribution database can be retrieved by running the following queries.

```
-- Collect the time associated with the transaction count into temp table
select t.publisher_database_id, t.xact_seqno,
max(t.entry_time) as EntryTime, count(c.xact_seqno) as CommandCount
into #results
FROM distribution..MSrepl_commands c with (nolock)
LEFT JOIN distribution.msrepl_transactions t with (nolock)
on t.publisher_database_id = c.publisher_database_id
and t.xact_seqno = c.xact_seqno
GROUP BY t.publisher_database_id, t.xact_seqno

-- Show each hour and number of commands per day:
SELECT publisher_database_id
, datepart(year, EntryTime) as Year
, datepart(month, EntryTime) as Month
, datepart(day, EntryTime) as Day
, datepart(hh, EntryTime) as Hour
-- , datepart(mi, EntryTime) as Minute
, sum(CommandCount) as CommandCountPerTimeUnit
FROM #results
GROUP BY publisher_database_id
, datepart(year, EntryTime)
, datepart(month, EntryTime)
, datepart(day, EntryTime)
, datepart(hh, EntryTime)
-- , datepart(mi, EntryTime)
-- order by publisher_database_id, sum(CommandCount) Desc
ORDER BY publisher_database_id, Month, Day, Hour

-- drop table #results
```

In the sample output below, a large batch of transactions were being replicated as a result of table updates causing slowdown in the Distribution Agent.

publisher_database_id	Year	Month	Day	Hour	CommandCountPerTimeUnit
2	2009	5	14	10	132
2	2009	5	14	11	656
2	2009	5	14	12	880
2	2009	5	14	13	4379
2	2009	5	14	14	152
2	2009	5	14	15	1478
2	2009	5	14	20	161
2	2009	5	14	21	145
2	2009	5	15	6	1700
2	2009	5	15	7	3672
2	2009	5	15	8	6266
2	2009	5	15	9	329
2	2009	5	15	10	5678715
2	2009	5	15	11	5637959
2	2009	5	15	12	5281732
2	2009	5	15	13	5020950
2	2009	5	15	14	1252
2	2009	5	16	11	732
2	2009	5	16	12	178
2	2009	5	16	13	725
2	2009	5	16	14	186
2	2009	5	16	16	72

The query below also provides overview of the commands stored in the Distribution database. Determine if the “large batch” of transactions correlate to high latency:

```
SELECT a.xact_seqno, a.entry_time,b.cnt
FROM (SELECT xact_seqno, COUNT(command_id) cnt
FROM distribution.msrepl_commands
GROUP BY xact_seqno ) b, distribution.msrepl_transactions a
WHERE a.xact_seqno = b.xact_seqno
ORDER BY a.entry_time
```

xact_seqno	entry_time	cnt
0x00036DA90000E2EB0038	2009-05-26 08:55:40.267	3131
0x00036D4F000021580091	2009-05-25 12:12:18.033	2974
0x00036DAF000085A800C3	2009-05-26 10:57:42.560	697488
0x00036DA100004C78003A	2009-05-26 06:02:23.770	595552
0x00036D510000A8D0004C	2009-05-25 13:46:55.743	463397
0x00036DB2000013480058	2009-05-26 12:34:13.997	1633
0x00036DA30000D3890089	2009-05-26 07:39:39.673	1633

### Scenario: High Statement CPU or I/O

A slow or overloaded database storage may be contributing to the long read times. Check the performance metrics on the ASC Troubleshooter for indications that the Distribution database is reaching I/O capacity.

Also check ASC Troubleshooter for queries that might have changed execution plans. High I/O and/or high CPU may be caused by out-of-date table statistics and thus bad query execution plans.

Check the system DMVs for details about duration and CPU of the Log Reader Writer-Thread. Low Duration and CPU but high Logical Reads may indicate a poor query execution plan caused by out of data table statistics.

```
-- dm_exec_query_stats for sp_MSget_repl_commands by top total_worker_time
SELECT TOP 25
st.text, qp.query_plan,
(qs.total_logical_reads/qs.execution_count) as avg_logical_reads,
(qs.total_logical_writes/qs.execution_count) as avg_logical_writes,
(qs.total_physical_reads/qs.execution_count) as avg_phys_reads,
qs.*
FROM sys.dm_exec_query_stats as qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
WHERE st.text like 'CREATE PROCEDURE sys.sp_MSget_repl_commands%'
ORDER BY qs.total_worker_time DESC
```

You may also run `sp_MSget_repl_commands` by yourself to gather execution details. This is more involved though and requires that you know the Log Sequence Number (LSN) of the current or a very recent transaction. You would usually see this in an -Output file; please see the "Investigation" section above for details.

```
USE distribution
GO
DBCC FREEPROCCACHE
DBCC DROPCLEANBUFFERS

SET STATISTICS PROFILE ON
SET STATISTICS IO ON
SET STATISTICS TIME ON


exec sp_MSget_repl_commands @agent_id=11, @last_xact_seqno=0x0000081a00000045000400000000

SET STATISTICS PROFILE OFF
SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

## Mitigation

- Large Batch of Transactions:

Large batch of transactions require heavy I/O requirements by the Distribution Agent Reader-Thread on the distribution database. Fast disk subsystem with Transaction log and database on separate drives/LUNs may help improve IO performance. If this will be an ongoing pattern, consider replicating the stored procedure EXECUTION instead of the RESULTS.

Please refer to the following article for further details: [Publishing Stored Procedure Execution in Transactional Replication](#) 

- High Statement CPU or I/O:

Check when the statistics were last updated on replication system tables; run Update Statistics if it is too old:

```
DBCC SHOW_STATISTICS(MSrepl_transactions,ucMSrepl_transactions)
DBCC SHOW_STATISTICS(MSrepl_commands,ucMSrepl_commands)

UPDATE STATISTICS MSrepl_transactions WITH FULLSCAN
UPDATE STATISTICS MSrepl_commands WITH FULLSCAN
```

Retrieve Distribution database settings and check if auto-update statistics is enabled:

```
exec sp_helpdb distribution --(look for IsAutoUpdateStatistics)
-- or:
select DATABASEPROPERTYEX('distribution','IsAutoUpdateStatistics') -- (returns 1 if enabled)
```

## Distribution Agent Writer Latency

When troubleshooting long Writer thread executions, the steps are the same as with any other, "normal" performance troubleshooting. The writer thread is executing stored procedures to perform the replication INSERT/UPDATE/DELETes on the Subscriber. If the writer thread is slow, it usually is the the execution of these procedures that you need to investigate. Depending on where the Subscriber is hosted, e.g. on-premise, in SQL Database, or in MI, you can use different tools to check the performance characteristics.

### Investigation

Check `distribution..MSdistribution_history` for the Distribution Agent performance statistics. You may see output like the following and interpret the `wait` times to see if either reader or writer thread is causing a delay:

```
<stats state="2" fetch="48" wait="384" cmds="1028" callstogetreplcmds="321">
  <sinclaststats elapsedtime="312" fetch="47" wait="284" cmds="1028" cmdsperssec="3.000000"/>
</stats>
```

If you notice `state="2"` events, it indicates that the agent is taking a long time to write changes to the destination. This is also reflected in a high wait time compared to the overall duration. In this example, the recent iteration = `sinclaststats` took 312 seconds overall, with 47 seconds spent on fetching data from the reader thread, and 284 seconds waiting for the write operations to complete. Commands per second = `cmdsperssec` is also very low.

A long writer thread execution might also show up in the normal `state="1"` entries. In the following example, `sinclaststats` took 764 seconds overall, of which the reader thread was waiting 505 seconds. The writer was busy all 764 seconds and waited only 50 seconds for data from the reader thread:

```
<stats state="1" work="1941" idle="0">
  <reader fetch="717" wait="1225"/>
  <writer write="1941" wait="134"/>
  <sinclaststats elapsedtime="764" work="764" cmds="1170730" cmdsperssec="1530.000000">
    <reader fetch="258" wait="505"/>
    <writer write="764" wait="50"/>
  </sinclaststats>
</stats>
```

Please see section "Log Reader and Distribution Agent Performance Statistics" above to get more details.

To get more details on the Subscriber performance in general, connect with SSMS to the Subscriber server. Then check the performance metrics on the SSMS Performance Dashboard reports for any bottlenecks. The stored procedures executed by the writer thread are named like `sp_MSins__articlename`, `sp_MSupd__articlename`, and `sp_MSdel__articlename`. If any of those procedures show up as involved in a bottleneck, you can derive the name of the target either from the procedure name or the SQL statement in the procedure.

Although it is not always desirable to interrupt the Distribution Agent, you might consider configuring the agent with an output file. The output file will record additional performance statistics that allows you to see

which steps are taking longest. The disadvantage is that you need to restart the agent, and if it was busy processing commands, it will have to redo the work after the restart. Also, if you provide an invalid file path, the agent will fail to start, thus causing additional delays.

Stop the Distribution Agent, add the "-Output" parameter with a valid Azure File Storage path, then restart the Agent. See article [Add verbose logging to Replication Agents in MI](#) for further details.

The output file breaks down the time for both the Reader (Fetch time) and the Writer (Write time) streams. In the sample output below, the Distribution Agent latency is occurring in the WRITER thread:

```
***** STATISTICS SINCE AGENT STARTED *****
Total Run Time (ms) : 18828 Total Work Time : 14110
Total Num Trans : 52 Num Trans/Sec : 3.69
Total Num Cmds : 52 Num Cmds/Sec : 3.69
Writer Thread Stats
Time Spent on Exec : 12063
Time Spent on Commits (ms): 422 Commits/Sec : 0.14
Time to Apply Cmds (ms) : 14110 Cmds/Sec : 3.69    <-----Writer thread
Time Cmd Queue Empty (ms) : 671 Empty Q Waits > 10ms: 2
Total Time Request Blk(ms): 671
Reader Thread Stats
Calls to Retrieve Cmds : 2
Time to Retrieve Cmds (ms): 92 Cmds/Sec : 565.22    <----- Reader thread
Time Cmd Queue Full (ms) : 5486 Full Q Waits > 10ms : 3
*****
```

In this output, the read operation completed in 92ms at a rate of 565 cmds/sec, while the WRITES are taking 14110ms (14 seconds) and only processing 3.69 cmds/sec. This example clearly shows the WRITE are slower than reads.

### Scenario: Long Execution Times in Insert/Update/Delete procedures

If you have identified one or more slow-running writer thread stored procedures, you can use the following DMV query to extract the query plan and performance details. Note the Where clause, which filters on the stored procedure name that you have identified:

```
-- dm_exec_query_stats for Subscriber procs by top total_worker_time
SELECT TOP 250
st.text, qp.query_plan,
(qs.total_logical_reads/qs.execution_count) as avg_logical_reads,
(qs.total_logical_writes/qs.execution_count) as avg_logical_writes,
(qs.total_physical_reads/qs.execution_count) as avg_phys_reads,
qs.*
FROM sys.dm_exec_query_stats as qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) as qp
WHERE st.text like '%sp_MSins_dboemployee%' -- sp_MSins_dbo<table name>%
--WHERE st.text like '%sp_MSupd_dboemployee%' -- sp_MSupd_dbo<table name>%
--WHERE st.text like '%sp_MSdel_dboemployee%' -- sp_MSdel_dbo<table name>%
ORDER BY qs.total_worker_time DESC
```

Or try to get it from QDS for past executions:

```
-- query to extract the sql text and query plan of specific queries
SELECT TOP 50
q.query_id, q.plan_id, query_hash, q.query_plan_hash, qt.query_sql_text, cast(q.query_plan as xml) query_plan
FROM
sys.query_store_plan AS q
JOIN sys.query_store_query AS qs ON q.query_id = qs.query_id
JOIN sys.query_store_query_text AS qt ON qt.query_text_id = qs.query_text_id
WHERE qt.query_sql_text like '%sp_MSins_dbo%' -- sp_MSins_dbo<table name>%
--qt.query_sql_text like '%sp_MSupd_dboemployee%' -- sp_MSupd_dbo<table name>%
--qt.query_sql_text like '%sp_MSdel_dboemployee%' -- sp_MSdel_dbo<table name>%
```

### Scenario: User Defined Triggers

Sometimes the customer has additional business logic implemented on the Subscriber database, where incoming changes are further processed by user-defined triggers on the Subscriber tables. The trigger executions are adding to the overall execution times and resource consumption of the writer thread stored procedures. They might also contribute to blocking scenarios, depending on how complex the business logic in the triggers is.

The easiest way to check for triggers is by connecting to the Subscriber database in SSMS and checking the table definition of the related article.

Another way is to look at the execution plan of the slow stored procedure and check for any plan branch that is not directly related to the target table. Please refer to the DMV queries mentioned in the scenario above for examples.

### Scenario: Too many indexes on the Subscriber tables

Sometimes the customer is using the Subscriber for reporting purposes. For speeding up report performance, it might make sense to add indexes for the Subscriber database that are not present on the Publisher database. Maintaining indexes costs performance, and these additional indexes may cause that a data change at the Subscriber is much slower than the corresponding change at the Publisher.

It is not straightforward to identify this scenario. Easiest is to ask the customer; otherwise you have to compare indexes on Publisher and Subscriber databases.

Sometimes you might see it from the publication script. Ask the customer to script out the publication, then check the `sp_addpublication` command and if the `@pre_snapshot_script` or `@post_snapshot_script` parameters are configured with additional script names.

### Scenario: Network Latency

This is very difficult to identify with PaaS environments, as there is no direct way of measuring the network latency and throughput. If you have already excluded other scenarios, and the Subscriber is remote to the Distributor, then the network might be the bottleneck.

Please check where the Subscriber is located, e.g. on-premise, in a different Azure region, on a different VNet, and how the routing is implemented.

The best option for a proper test might be to deploy an Azure VM into the same VNet at the Managed Instance, then perform network tests towards the Subscriber SQL Server. You might have to include the Azure Networking support team in the analysis.



## Scenario: Changes not replicated as default parameterized Stored Procedures

If the articles in the publication are created with default values, all changes are replicated using parameterized stored procedures that are named like `sp_MSins_%/sp_MSupd_%/sp_MSdel_%`.

But the customer always has the option to shoot themselves into the foot by changing the behaviour:

- using un-parameterized stored procedure calls, or
- replacing the default stored procedures with custom stored procedures, or
- replacing the default stored procedures with plain SQL statements.

Neither of these options is optimal, opening the door for all kinds of performance nastiness, and may cause severe delays on the writer thread.

You can identify this by checking the article definitions in the Publisher database:

```
use <publisher database>
select name, ins_cmd, upd_cmd, del_cmd, status, ins_scripting_proc, upd_scripting_proc, del_scripting_proc
from dbo.sysarticles
```

The result from a set of default articles looks similar to this:

name	ins_cmd	upd_cmd	del_cmd
Article1	CALL [dbo].[sp_MSins_dboArticle1]	SCALL [dbo].[sp_MSupd_dboArticle1]	CALL [dbo].[sp_MSdel_dboArticle1]
Article2	CALL [dbo].[sp_MSins_dboArticle2]	SCALL [dbo].[sp_MSupd_dboArticle2]	CALL [dbo].[sp_MSdel_dboArticle2]

This example above is the preferred configuration:

- ins\_cmd/upd\_cmd/del\_cmd show the expected sp\_MSins/sp\_MSupd/sp\_MSdel procedure calls
- status=57 includes value 24 for using parameterized statements (57=1+24+32)
- the %\_scripting\_proc columns are all NULL, indicating that no custom procs have been configured

But you might see the following variations:

- status includes value 8 instead of 16 or 24, e.g. is 41=1+8+32
- ins\_cmd/upd\_cmd/del\_cmd show "SQL" instead of the procedure names
- the %\_scripting\_proc columns have custom stored procedure names

Reference: [sysarticles](#) 

```
1 = Article is active.
8 = Include the column name in INSERT statements (implying that 'parameters' option is not enabled).
16 = Use parameterized statements.
24 = Both include the column name in INSERT statements and use parameterized statements.
```

## Mitigation

- Long Execution Times: As with other performance cases, high duration with low CPU and I/O indicates blocking of the writer thread on the Subscriber. Use the normal tools like Activity Monitor, SSMS Performance reports, blocker script etc. to investigate the source of the blocking.
- User Defined Triggers: Use standard query performance troubleshooting techniques to improve the execution of the user-defined triggers. Discuss with the customer if the triggers are really necessary or can be replaced with some other implementation.
- Too many indexes on the Subscriber tables: Discuss with the customer if the additional indexes are really necessary. You might have to use standard query performance troubleshooting techniques to improve the execution of the Subscriber report queries.
- Network Latency: Review the network topology between Distributor and Subscriber. Involve Azure Networking support team in the analysis.
- Changes not replicated as default parameterized Stored Procedures: Discuss with customer why the default configuration has been changed, and if it could be changed to the default. Configuring the Subscriber with auto-parameterization might lessen the impact.

The article status can be updated on-demand using the following replication stored procedure - it does not cause a reinitialization of the subscriptions:

```
EXEC sp_changearticle @publication = N'<pub name>', @article = N'<article name>', @property = 'status', @value = 'parameters'
```




Reference: [sp\\_changearticle](#) 

## Design options for improving replication performance

There are several public articles available that might help you and your customers with steps to improve replication performance.

Especially the options regarding using SubscriptionStreams and the agent parameters for batching can provide short-term relief on performance bottlenecks.

References:

- [Enhance Transactional Replication Performance](#) 
- [Enhance General Replication Performance](#) 
- [Publishing Stored Procedure Execution in Transactional Replication](#) 

Please leave comments below if you have any suggestions for improving this article!

**How good have you found this content?**



-