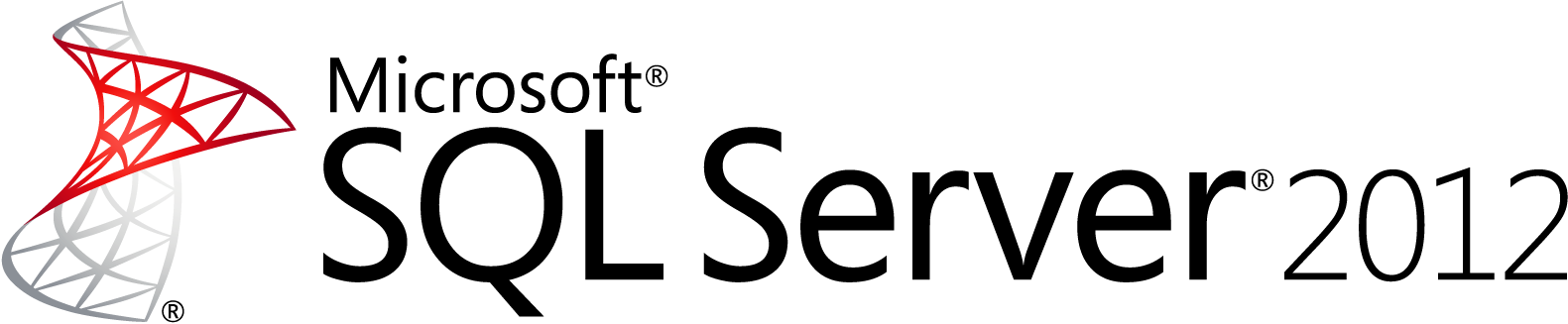
SQL Server Technical Article



**AlwaysOn: Offloading Read-Only Workloads to Secondary Replicas**

**Writer:** Sunil Agarwal

**Technical Reviewers:** Kevin Farlee, Sanjay Mishra, Cephas Lin, Mike Weiner, Prem Mehra

**Published:** July 2012

**Applies to:** SQL Server 2012

**Summary:**

In Microsoft SQL Server 2012, the AlwaysOn Availability Groups technology can address a common requirement of your existing high availability solution: utilization of secondary servers to result in high ROI of the investment in hardware. With AlwaysOn Availability Groups, these otherwise inactive secondary servers can be utilized as readable secondary replicas. This feature enables you to optimize your hardware utilization, thus achieving higher ROI for your hardware. Furthermore, it can help you to improve performance on the primary system by offloading your read-only workload to readable secondary replicas. This white paper describes how to configure AlwaysOn Availability Groups for offloading reporting workload to a secondary replica. It also describes the factors that can affect the high availability and the performance of your reporting workload.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious.  No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.

Contents

[Introduction 4](#_Toc327245351)

[Offloading Reporting Workload in SQL Server 2008 R2 4](#_Toc327245352)

[Value Proposition of AlwaysOn Readable Secondary Replicas 5](#_Toc327245353)

[Setting Up Readable Secondary Replicas 5](#_Toc327245354)

[Impact of Running Reporting Workload on a Secondary Replica 7](#_Toc327245355)

[Impact on High Availability 7](#_Toc327245356)

[Impact of Reporting Workload Running Under Snapshot Isolation Level 9](#_Toc327245357)

[Troubleshooting REDO Blocking 12](#_Toc327245358)

[Impact on Primary Workload 13](#_Toc327245359)

[Impact on Reporting Workload 15](#_Toc327245360)

[Data Latency 15](#_Toc327245361)

[Query Plans 15](#_Toc327245362)

[Connecting to Secondary Replicas 20](#_Toc327245363)

[Summary 20](#_Toc327245364)

# Introduction

The AlwaysOn Availability Groups technology in Microsoft SQL Server 2012 includes a feature called readable secondary replicas. It enables you to take advantage of your existing investment in high availability hardware for offloading read-only workloads such as reporting to one or more secondary replicas. Offloading read-only workloads to secondary replicas frees up resources on the primary replica to achieve higher throughput for an OLTP workload. At the same time, it allows resources on secondary replicas to deliver higher performance on read-only workloads. So it is a win-win situation for both primary and reporting workloads.

Before proceeding further, it is useful to look at the technology choices for offloading read-only workload that are available in versions earlier than SQL Server 2012. This review provides insight into the unique value proposition of readable secondary replicas.

## Offloading Reporting Workload in SQL Server 2008 R2

SQL Server 2008 R2 and many earlier releases offer the following three high availability choices to customers who want to offload a reporting workload to secondary replicas. Each technology has strengths and weaknesses:

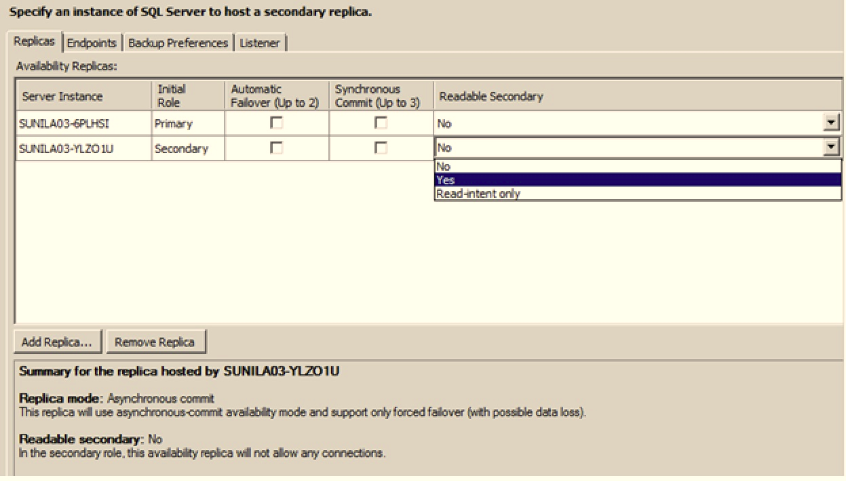
* **Database mirroring:** The mirror database in the database mirroring configuration is not readable, but you can create a database snapshot on the mirrored database, which is readable. The read-only workload can then run against the database snapshot. This approach has the following challenges:
  + The name of the database snapshot is different from the name of the database in the database mirroring configuration. If your application has a hard-coded database name, you need to make modifications in order to connect to the snapshot.
  + The database snapshot is a static view of the data at the time the snapshot was taken. If an application needs to access more recent data, you must create a new database snapshot with a new name unless you drop the old database snapshot. In other words, near real-time access to data is difficult, if not impossible, to achieve if you are using a database snapshot.
  + A database snapshot employs copy-on-write operations. These operations can add significant overhead if there are multiple database snapshots. Also, queries that run on a database snapshot incur more random I/O. Together, these issues can cause significant performance degradation.
* **Log shipping:** With log shipping, you can run a reporting workload on the log shipping target node, but the data latency incurred by the reporting workload depends on the frequency of the transaction log restore. However, if the secondary database is open for reporting workload, the log backups cannot be restored. This can be management challenge because you must choose between high availability and the latency incurred by a reporting workload. If you reduce the frequency of the log restores, both the data latency for the reporting workload and the recovery time objective (RTO) are negatively affected. If you increase the frequency of the transaction log restore, you must disconnect all users on the secondary database before the restore. That does not work for many scenarios, especially if you have long-running queries, because they may never run to completion due to RTO constraints.
* **Replication:** Transactional replication can be used as a solution for offloading read and reporting workloads. A few key benefits of replication are that customers can create reporting-workload-specific indexes and filter the dataset on the subscriber database. The challenges here include the following: all tables require a primary key, which may require schema changes (for example, unique key), and replication is not suitable for high transaction throughput. There can be significant latency between publisher and subscriber, particularly under large batch jobs and high transaction volumes.

## Value Proposition of AlwaysOn Readable Secondary Replicas

Readable secondary replicas address the challenges of previous high availability solutions, with the exception of the ability to create reporting workload-specific indexes similar to the ones you use in transactional replication. With a readable secondary replica, you can run the read workload concurrently with the recovery thread that is applying the transactional logs from the primary. This concurrency allows a reporting workload to access the data changes live as they occur on the primary replica and are subsequently applied by the REDO thread on the secondary replica. The reporting workload is run without requiring any changes, because the database on the secondary replica is same as the database on the primary.

# Setting Up Readable Secondary Replicas

To set up a readable secondary replica, you first create an availability group. Then you add replicas. The easiest way to do that is to use the Availability Group Wizard. In the Availability Group Wizard, you enable one or more secondary replicas for read-only workloads as shown in the following picture.



You can choose either **Yes** or **Read-intent only** options. You can run the read workload on the secondary replica with either option, but there are few differences:

* **Yes** option: Supported TDS clients can connect to the secondary replica explicitly to run the reporting workload. The client is responsible for ensuring that it is connecting to readable secondary, because the roles of replicas can change in case of failover. The key benefit of this option is that older clients can run reporting workloads on the readable secondary.
* **Read-intent-only** option: Only connections that have the property ApplicationIntent set to ReadOnly are accepted. The word *intent* indicates that you want to use the connection as read-only; it does not prevent read/write connections. It is still possible to connect using a read/write application if the ApplicationIntent option is set to ReadOnly, but the application fails on the first DML or DDL operation. This option allows clients to automatically connect to an available readable secondary, and you can use it to prevent read workloads from running on the primary replica. For more information about how to use this setting, see *Connecting to Secondary Replicas* later in this white paper.

After you configure the availability group, you can connect to the secondary replica either explicitly or by using an availability group listener (also known as a virtual network name, or VNN) and then run your reporting workload. For more information about options for readable secondary connections, see [Configure Connection Access on an Availability Replica](http://msdn.microsoft.com/library/hh213002(SQL.110).aspx) in Books Online.

# Impact of Running Reporting Workloads on a Secondary Replica

When you offload a reporting workload to a secondary replica, a common expectation and experience is that the primary workload performs better because it does not have to complete for resources with a reporting workload. Similarly, the reporting workload performs better because it has access to more resources on the secondary replica. The following sections look at the impact of running a reporting workload on:

* High availability
* Primary replica
* Reporting workload

For more information about read-only workloads on secondary replicas, see [Readable Secondary Replica](http://msdn.microsoft.com/library/ff878253(SQL.110).aspx) in Books Online.

## Impact on High Availability

The primary goal of AlwaysOn technology is to provide high availability for mission-critical workloads. The ability to offload reporting workloads and database/transaction log backups onto secondary replicas is useful, but only if it does not compromise high availability. When you evaluate a high availability solution, you take two measures, RPO and RTO, into account. Let’s look at each measure to understand the implications of running the reporting workload on the secondary replica:

* **Recovery point objective (RPO):** RPO is the potential data loss (in time units) that can occur if the primary replica fails. If the secondary replica chosen as the new primary is configured with synchronous commit mode, data loss is very unlikely, whether the readable secondary replica has been enabled. If the secondary replica was set in asynchronous commit mode, there is no *additional* impact on the RPO (although you can lose data if you fail over to an asynchronous replica) due to running the reporting workload on the secondary replica. The reason is that the reporting workload does not interfere with the log transfer path from the primary replica to the log buffer and its ability to harden the transaction on the secondary replica, assuming your log files are mapped to their own storage.
* **Recovery time objective (RTO):** RTO in this case is the time needed to bring the database online after the primary fails. When the database restarts after a failover, the REDO thread needs to apply the transaction log records from its current position to the end of the transaction log. The farther the REDO thread has fallen behind in applying the transaction, the longer it takes to bring the database online (that is, the RTO is larger).

A read-only workload can impact the RTO in the following ways:

* + As part of applying log records on the secondary replica, the REDO thread reads the log records from the log disk, and then for each log record it accesses the data pages to apply the log record. The page access can cause physical I/O if the page is not already in the buffer pool. If the reporting workload is I/O bound, it competes for I/O resources with the REDO thread, potentially slowing it down. You can use Resource Governor to control CPU cycles that are used by the reporting workload to indirectly control the I/O cycles taken, to some extent. For example, if your reporting workload is consuming 10 percent of CPU but the workload is I/O bound, you can use Resource Governor to limit CPU resource usage to 5 percent to throttle read workload, which minimizes the impact on I/O.
  + The REDO thread can be blocked by the reporting workload. A blocked REDO cannot apply log records until it is unblocked, which can lead to unbounded impact on the RTO. This blocking can happen if the REDO thread is executing DML or DDL operations. Here are some examples of each type of blocking with descriptions of how the use of readable secondary replicas solves the problem:
    - **DML operations:** Consider a reporting workload that runs under repeatable read isolation level. This reporting workload locks row R1 with a shared (S) lock. Now, on the primary replica, another transaction updates row R1. Because there is no shared (S) lock on row R1 on the primary replica, the transaction on the primary node completes successfully. Now, the log records for the update transactions are applied by the REDO thread on the secondary replica. To update row R1, the REDO thread must acquire an exclusive (X) lock, but the thread is blocked by concurrent reporting workload. This type of blocking between DML and reporting (that is, between writers and readers) is very common. The use of readable secondary replicas eliminates this blockage by mapping isolation levels used in the reporting workload to the snapshot isolation level. The next section covers this mapping in more detail.
    - **DDL operation:** Consider a reporting workload that runs under snapshot isolation and accesses table T1. Although SQL Server does not take any locks on the data rows, it does take a schema stability (Sch-S) lock on table T1. Concurrently, a DDL transaction on the primary replica rebuilds the clustered index online on table T1. You may recall that SQL Server acquires a short-term schema-modification (Sch-M) lock to update the schema information. The index is built successfully without any blocking because there is no reporting thread blocking the schema-modification (Sch-M) lock. However, on the secondary replica, the REDO thread is blocked. Although this is something to consider, because DDL operations are not a common occurrence in a production environment, the DDL blocking is generally infrequent. To help recognize this scenario, an Extended Event (XEvent) is raised. If enabled, the XEvent can be used to alert the administrator. For more information, see Troubleshooting REDO blocking later in this white paper.

### Impact of a Reporting Workload That Runs Under Snapshot Isolation

The first thing to understand is that if you do not enable a secondary replica for read-only workload, no snapshot isolation mapping is needed, and therefore no additional snapshot isolation level related overhead occurs on the primary replica. Things get interesting when you enable the secondary replica for read-only workloads. You may recall that both Snapshot Isolation (SI) and Read Committed Snapshot Isolation (RCSI) are based on row versioning[[1]](#footnote-1). When a row is modified, its previous version is saved in the version store backed by **tempdb** and a 14-byte pointer is set from the modified row to the versioned row. If the data row is modified again, the process is repeated. The key point to note here is that newly inserted or modified data rows always have a 14-byte overhead. There are four possible scenarios to consider:

* SI and RCSI are not enabled on the primary replica and the secondary replica not enabled for read.
* SI and RCSI are not enabled on the primary replica but the secondary replica is enabled for read.
* SI and/or RCSI are enabled on the primary replica but the secondary replica not enabled for read.
* SI and RCSI are enabled on the primary and the secondary replica is enabled for read.

For more information about the row versioning overhead for each scenario, see the sections later in this paper.

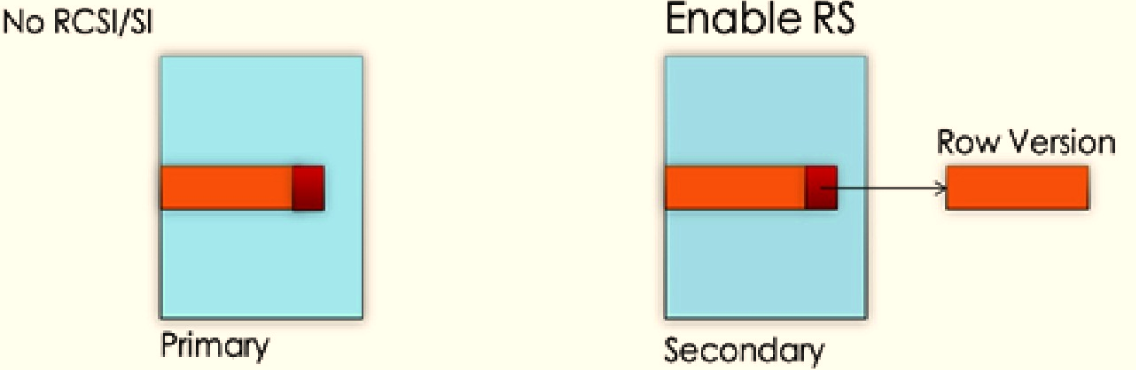
#### Secondary Replica Not Enabled for Read Workload

In this case, SI and/or RCSI are not enabled on the primary replica and the secondary replica is not enabled for read workload. As shown in the following picture, there is no row versioning overhead on either the primary replica or the secondary replica.

[[2]](#footnote-2)

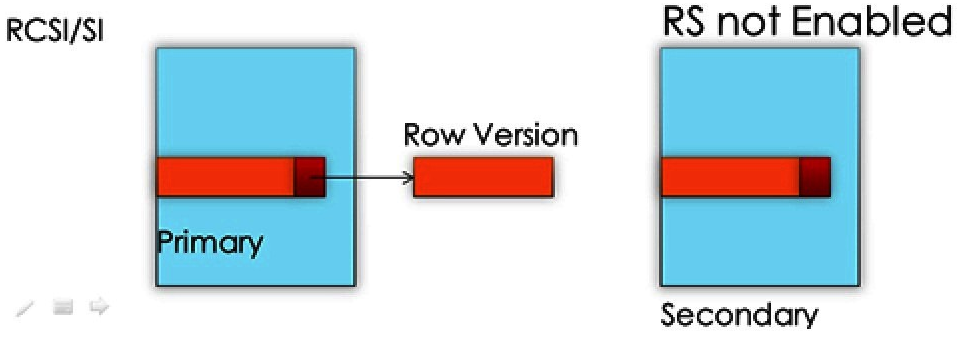
#### Secondary Replica Enabled for Read Workload

In this case, SI and/or RCSI are not enabled on the primary replica, but the secondary replica is enabled for read workload. There are two interesting points to note here. First, the row version is only generated on the secondary replica; because RCSI or SI is not enabled on the primary replica, there is really no need to create row versions there. Second, the row versions need to be generated on the secondary replica, which means that the 14-byte overhead needs to be added to the new and modified rows on the primary, because the primary and secondary replicas must be physically identical. Existing rows that are not modified do not incur the 14-byte overhead. The following picture shows the 14-byte overhead on the primary replica and the generation of the row version on the secondary replica.



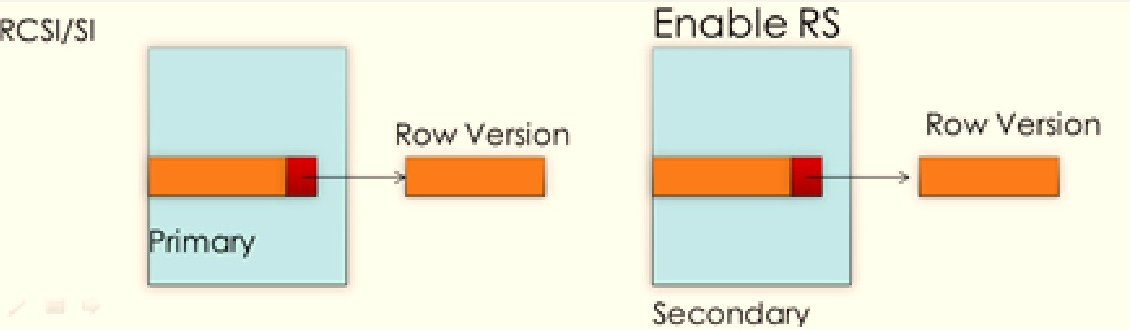
#### Primary Replica RCSI or SI Enabled, Secondary Replica Not Read Enabled

In this instance, SI and/or RCSI are enabled on the primary replica but the secondary replica is not enabled for read workload. This case is a bit simpler because the 14-byte versioning overhead is already added to the data rows on the primary replica independent of the status of secondary replica. As shown in the following picture, if the secondary replica is not enabled for read workload, there is still a 14-byte overhead on the rows on the secondary replica, but there is no row version generation on the secondary because the read workload has not been enabled.



#### Primary Replica RCSI or SI Enabled, Secondary Replica Read Enabled

In this case, SI and/or RCSI are enabled on the primary replica, and the secondary replica is enabled for read workload. This case is similar to the previous configuration except that row versions must also be generated on the secondary replica. The following picture shows the 14-byte overhead in the data/index row and the row version.



There is no change or additional guidance for managing and provisioning for row versioning in this configuration compared to what we recommend for snapshot isolation or RCSI. For more information about database isolation levels, see [Isolation Levels in the Database Engine](http://msdn.microsoft.com/en-us/library/ms189122.aspx) in Books Online.

The following table summarizes the scenarios.

|  |  |  |
| --- | --- | --- |
| **AlwaysOn** | **Primary Replica** | **Secondary Replica** |
| Primary: SI/RCSI - No  Secondary: Read – No | No row versioning overhead | No row versioning overhead |
| Primary: SI/RCSI – No  Secondary – Read - Yes | 14-byte overhead: Yes  Row versions: No | 14-byte overhead: Yes  Row versions: Yes |
| Primary: SI/RCSI – Yes  Secondary: Read - No | 14-byte overhead: Yes  Row versions: Yes | 14-byte overhead: Yes  Row versions: No |
| Primary: SI/RCSI – Yes  Secondary: Read – Yes | 14-byte overhead: Yes  Row versions: Yes | 14-byte overhead: Yes  Row versions: Yes |

### Troubleshoot REDO Blocking

As described earlier, when the REDO thread executes transaction log records for DDL operation, it can be blocked by a reporting workload. When this happens, a ‘lock\_redo\_blocked’ Extended Event is generated. Additionally, you can query the DMV **sys.dm\_exec\_request** on the secondary to find out which session is blocking the REDO thread, and then you can take corrective action. The following query shows that REDO thread (session-id 38) is blocked, waiting to acquire schema modification (Sch-M) lock, on user session 51

-- show the REDO thread blocked

select session\_id, command, blocking\_session\_id, wait\_time, wait\_type, wait\_resource

from sys.dm\_exec\_requests where session\_id = 38

38 DB STARTUP 51 4909746 LCK\_M\_SCH\_M OBJECT: 6:245575913:0

You can also query the following DMV to see how far the REDO thread has fallen behind. The query output shows that the last ‘redone’ LSN is 138000000058300002 and the received LSN on the secondary is 138000000082500001, indicating that REDO is getting behind.

select recovery\_lsn, truncation\_lsn, last\_hardened\_lsn, last\_received\_lsn,

last\_redone\_lsn, last\_redone\_time

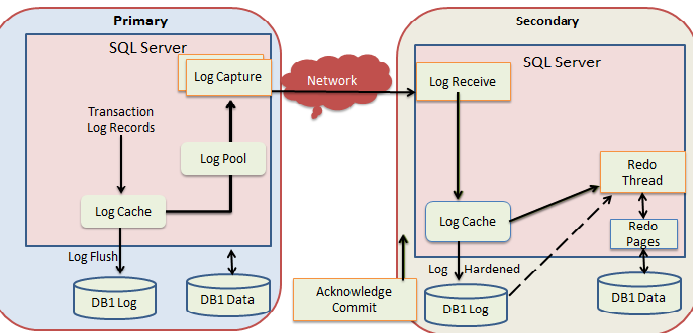
from sys.dm\_hadr\_database\_replica\_states



## Impact on Primary Workload

When a secondary replica is enabled for running reporting workloads, any changes to the rows done as part of DML operation start incurring a 14-byte overhead, as explained earlier. Note that no changes to the size of rows need to be made to existing rows. This 14-byte overhead occurs only when an existing row is updated or deleted or when a new row is added. This is very similar to the impact of enabling RCSI/SI on the primary, except that no row versions need to be generated on the primary replica. The extra 14 bytes can lead to more page splits as the size of the row or rows is increased.

However, the reporting workload does not affect the transactional throughput on the primary replica. When a transaction is run on the primary replica, the transaction log records are written to the log buffer and at the same time are sent to the log pool to be sent to the secondary replica (in this example there is only one secondary replica, but the same logic holds for multiple replicas) as shown in the following picture.



If the secondary replica is configured in asynchronous commit mode, there is no impact on the transaction response time on the primary. However, if the secondary replica is configured in synchronous commit mode, the primary replica only commits a transaction when the commit log record has been hardened on the secondary replica (at this point it can send an acknowledgement to primary replica). A delay in the acknowledgement from the secondary replica adds to the latency of the transaction.

This leads to the question: “Do read workloads running on the secondary replica impact the acknowledgement (ACK) for the transaction commit?” The answer is that this is unlikely. In the secondary replica in the preceding picture, there are essentially two background threads: one receives the log record over the network and the other hardens that log record. SQL Server gives priority to background threads over user threads (including the ones that are running read workload). This means that at least from the CPU perspective, a read workload cannot delay the ACK. An I/O intensive read workload could slow down the transaction log write, but this would only happen if the data and the transaction log were to share the same physical disk. In most production deployments, transaction log disks are not shared with data disks, so it is a nonissue. However, a network bottleneck can add to the latency of the transaction, but in that case it is unrelated to read workload. In summary, in a well-configured and well-managed system, it is unlikely that the read workload on the primary will add to the transactional latency.

## Impact on Reporting Workload

### Data Latency

The reporting workload running on the secondary replica will incur some data latency, typically a few seconds to minutes depending upon the primary workload and the network latency. The data latency exists even if you have configured the secondary replica to synchronous mode. While it is true that a synchronous replica helps guarantee no data loss in ideal conditions (that is, RPO = 0) by hardening the transaction log records of a committed transaction before sending an ACK to the primary, it does not guarantee that the REDO thread on secondary replica has indeed applied the associated log records to database pages. So there is some data latency. You may wonder if this data latency is more likely when you have configured the secondary replica in asynchronous mode. This is a more difficult question to answer. If the network between the primary replica and the secondary replica is not able to keep up with the transaction log traffic (that is, if there is not enough bandwidth), the asynchronous replica can fall further behind, leading to higher data latency. In the case of synchronous replica, the insufficient network bandwidth does not cause higher data latency on the secondary but it can slow down the transaction response time and throughput for the primary workload.

If your reporting workload cannot tolerate any data latency, you must run it on the primary replica. The good news is that generally most reporting workloads can tolerate some data latency and therefore can safely be migrated to secondary replica.

### Query Plans

One of the goals of moving the reporting workload to a secondary replica is comparable or even increased performance as it does not need to compete for resources with primary workload. Ideally, both the primary workload and the reporting workloads should perform better after the move because they do not need to compete for resources on the same SQL Server instance.

It is important for the latest column statistics to be available on the secondary replica to generate an optimized query plan that is comparable to the query plan for the reporting workload on the primary replica. The following sections describe the column statistics challenge and how it is addressed.

#### The Statistics Challenge

The secondary replica is accessed by the reporting workload in read-only mode, which prevents auto-update on the statistical information needed by the query optimizer. Let’s consider an example of a read-only database to illustrate the impact on stale statistics in a read-only database.

**Example**

use test; go

create table t1 (c1 int, c2 int, c3 char (50)); go

create nonclustered index t1\_nci on t1(c1); go

--insert 1 row

insert into t1 values (1,1,'hello')

--do a simple select and show the query plan

select c2 from t1 where c1 between 1 and 20

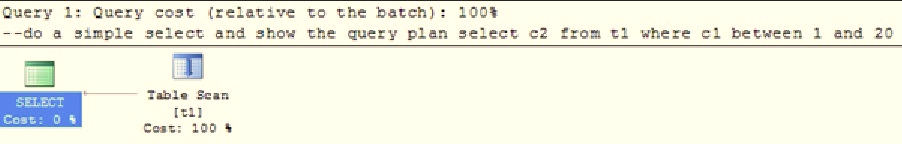
-- show the statistics

dbcc show\_statistics('t1', 't1\_nci')

Here is the output. The statistics now show that there is now one row. Note that these statistics are auto-created by SQL Server when the query is run.



Another interesting thing to note here is to see the query plan. Note that the optimizer chooses a table scan because it has determined that if there is only one row; it is cheaper to scan the table than it is to use the nonclustered index.



The next query inserts 10,000 rows into table T1 and then marks the database as read-only.

declare @i int = 0

while (@i < 10000)

begin

insert into t1 values (@i, @i+100000, 'sunil')

set @i = @i + 1

end

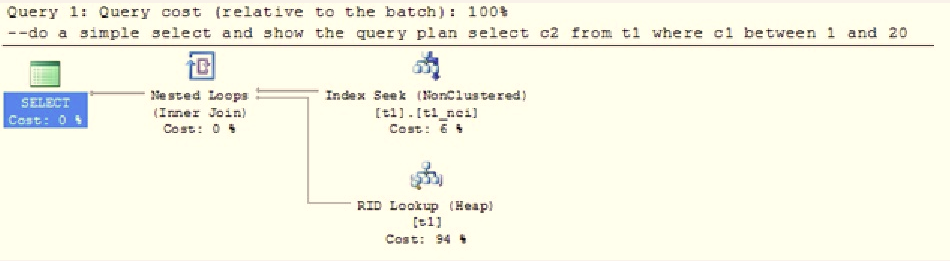
ALTER DATABASE [test] SET READ\_ONLY WITH NO\_WAIT

GO

Because the database is now marked as read-only, no statistics changes can take place. If you run the following query, the query optimizer still chooses a table scan because it is aware of only one row in the table. As you can imagine, a table scan would cause this query to run much slower. Ideally, the optimizer should choose the index scan.

select c2 from t1 where c1 between 1 and 20

Mark this database as read/write (R/W) and run the query again. Because the database is R/W, SQL Server updates the stale statistics and correctly chooses the plan utilizing the index as shown here.



Now, let’s observe what happens on AlwaysOn secondary replica. The example remains the same except that you now execute the query on the secondary replica after inserting 10,000 rows on the primary replica. Because the database on the secondary replica is only accessible in read-only mode, there are two possibilities:

* *Stale statistics*: The statistics on table t1 column c1 are stale, indicating that there is only one row (assuming you ran the query on the primary replica after inserting one row). Any statistics created on primary replica are created on the secondary replica by the REDO thread.
* *Missing statistics:* The statistics are missing completely if you did not run a query on the primary replica that used a predicate on table t1 column c1.

The next section describes how this issue is addressed in AlwaysOn.

#### Solution to Statistics Problem in AlwaysOn Readable Secondary

The first thing to note is that any statistics created on the primary replica are automatically available on the secondary replica for usage. The challenge is in allowing missing statistics to be created or stale statistics to be updated on the secondary replica. The short answer is that this is not possible because it violates the rule that the primary and secondary database must be physically identical. However, statistics on an object can be created and re-created using the data in the table. Based on this fact, temporary statistics are created and stored in **tempdb**. This change guarantees that up-to-date statistics are available on the secondary replica just like they are on the primary replica for the query optimizer. The implication of creating temporary statistics is that these statistics can be lost if SQL Server is restarted, but this is not a true data-loss situation because, as noted earlier, these statistics can be re-created at a relatively low cost by querying the underlying objects. Similarly, the temporary statistics are removed when a primary replica fails over.

To illustrate how temporary statistics on the readable secondary interact with statistics available through primary replica, consider table T1 with three columns C1, C2, and C3. For this discussion, the type of column does not matter. C1prim and C1sec represent the statistics on column C1 that were created on the primary and secondary replicas, respectively.

The following table summarizes the various interactions.

|  |  |  |
| --- | --- | --- |
| **Action** | **Primary Replica** | **Secondary Replica** |
| Query on the secondary with a predicate on C1. | No statistics on C1 exist. | C1sec is created in **tempdb**. |
| Query on the primary with a predicate on C1. | C1prim is created. | C1prim is created when the log for the statistics is processed. At this time, both C1sec and C1prim exist on secondary replica but C1prim is the most recent, and the optimizer will use it. At this time C1sec is not useful and you can explicitly drop it. |
| Memory pressure forces T1 out of the cache. |  | C1sec is removed from the cache but it still persists in **tempdb**. |
| Insert a large number of rows in T1 such that the auto-stat threshold is crossed. Now query on the secondary with a predicate on C1. |  | C1sec is refreshed. |
| Query on the primary with a predicate on C2. | C2prim is created | C2prim is created on when the log for the statistics is processed. |
| Insert a large number of rows in T1 such that the auto-stat threshold is crossed. Now query on the secondary with a predicate on C2. |  | C2sec is created. At this time, both C2sec and C2prim exist on the secondary replica but C2sec is more recent and the optimizer will use it. |
| Perform a DDL operation on table T1. |  | Cached metadata for T1 is deleted and as part of this process C1sec and C2sec are dropped. |

A new column **is\_temporary** is added to the catalog sys.stats to indicate whether a statistic is temporary or not. You can drop the temporary statistics. Also, when temporary stats are created or updated on the secondary replica, an Extended Event (XEvent) *AutoStats* is generated (assuming you have enabled it). This is the same XEvent that gets generated in SQL Server for statistics updates and creation.

# Connecting to Secondary Replicas

You can connect to a secondary replica if it has been enabled for read-only workloads. You can choose to connect directly to the SQL Server instance for the secondary replica and can run your reporting workload. However, if the secondary replica assumes the role of primary replica in case of a failover, you may unintentionally run the reporting workload on the *new* primary. To prevent this, you have two choices:

* You can connect to the secondary replica using node\instance-name: Configure the primary replica to PRIMARY\_ROLE (ALLOW\_CONNECTIONS = READ\_WRITE) and use ‘ApplicationIntent=ReadOnly’ on your connection string for reporting workload. In this case, if you mistakenly connect your reporting workload to a physical node that is already running as a primary replica, the connection is denied.
* You can connect to the secondary replica with ReadOnly routing configured: Connect the reporting workload using the listener, which is a virtual network name (VNN), with Application Intent = ReadOnly specified in the connection string. This setting tells SQL Server that you intend to connect to a secondary replica for reporting workload, and so SQL Server transparently connects the workload to one of the secondary replicas that has been configured to run reporting workload.

**Note**: When the connectivity between the replicas is broken, the secondary replica cannot be used to offload read workload.

# Summary

Many customers have successfully deployed their reporting workloads on the secondary replica to achieve higher throughput both for primary and report workloads. Your reporting workload can run on a secondary replica with a comparable query plan without requiring any changes. For more information about AlwaysOn and readable secondary replicas, see <http://blogs.msdn.com/b/sqlalwayson/>.

1. For more information about snapshot isolation, see <http://msdn.microsoft.com/en-us/library/ms189050.aspx>. [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)