

# Online Index Operations

SQL Server Technical Article

Writers: Alexis Boukauvalas, Gang He, Lyudmila Fokina, Mirek Sztajno, Sameer Verkhedkar, Sanjay Mishra, Wey Guy, Gail Erickson

Technical Reviewer: Sunil Agarwal, Mike Ruthruff, Srikumar Rangarajan

Published: March 2007

Applies To: SQL Server 2005 Enterprise Edition

Summary: Introduced in SQL Server 2005 Enterprise Edition, the online index feature provides a powerful way to perform maintenance operations such as rebuilding or creating indexes in a production system without sacrificing DML concurrency. This paper provides a detailed discussion of the index process and provides guidelines and best practices for implementing this feature in a production environment.

### Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

 2007 Microsoft Corporation. All rights reserved.

Microsoft, Windows, and Windows Serverare either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

### Table of Contents

[Introduction 1](#_Toc160958527)

[Data Structures and Concepts 1](#_Toc160958528)

[Overview of the Online Index Build Algorithm 1](#_Toc160958529)

[Preparation Phase 3](#_Toc160958530)

[Build Phase 5](#_Toc160958531)

[Mapping Index and Clustered Index Operations 5](#_Toc160958532)

[Impact of Nonclustered Indexes 6](#_Toc160958533)

[Final Phase 6](#_Toc160958534)

[Capacity Planning and Concurrency Considerations for Index Create and Rebuild Operations 7](#_Toc160958535)

[Temporary Storage Requirements 7](#_Toc160958536)

[Measuring Temporary Disk Space Usage 8](#_Toc160958537)

[Determining the Amount of Temporary Space Used 9](#_Toc160958538)

[Example 11](#_Toc160958539)

[Observations 12](#_Toc160958540)

[Online Index Best Practices 13](#_Toc160958541)

[Determine the Index Operation Needed for Your Environment 13](#_Toc160958542)

[Use ONLINE and SORT\_IN\_TEMPDB Effectively 14](#_Toc160958543)

[Maximize tempdb Space and Throughput 14](#_Toc160958544)

[Manage Transaction Log Space 15](#_Toc160958545)

[Transaction Log Test Measurements 15](#_Toc160958546)

[Control Parallelism and Memory Resources 16](#_Toc160958547)

[Parallelism and Partitioned Indexes 16](#_Toc160958548)

[Conclusion 17](#_Toc160958549)

[Appendix A: Performance Study 17](#_Toc160958550)

[Test Cases and Set Up Details 17](#_Toc160958551)

[Performance Without Concurrent Activity 18](#_Toc160958552)

[Test1: Creating a Nonpartitioned Unique Clustered Index on a Heap 18](#_Toc160958553)

[Test 2: Creating a Nonpartitioned Nonclustered Index on a Clustered Index 20](#_Toc160958554)

[Test 3: Creating a Partitioned Clustered Index 21](#_Toc160958555)

[Performance with Concurrent Activities 22](#_Toc160958556)

[Concurrent Workload Impact on Online Index Operation 22](#_Toc160958557)

[Online Index Operation Impact on Concurrent Workload 23](#_Toc160958558)

[Test Case Summary 23](#_Toc160958559)

[Appendix B: Diagnostics 24](#_Toc160958560)

[Appendix C: Log Measurement Script 24](#_Toc160958561)

[Appendix D: Online Index Limitations 25](#_Toc160958562)

#### Introduction

Introduced in Microsoft® SQL Server™ 2005 Enterprise Edition, the online index feature provides a powerful way to perform maintenance operations such as rebuilding or creating indexes in a production system without sacrificing DML concurrency. Users are not blocked from querying and updating the underlying table during the index operation.

This white paper is targeted to database administrators who want to use the online index feature of SQL Server 2005 Enterprise Edition to increase server availability in their production environment and need to understand the advantages and disadvantages of using this feature.

In this paper, we describe the online index process in detail to provide insight into how the process works. We include capacity planning guidelines for estimating the temporary space requirements required for online operations and describe the tools you can use to estimate and measure the amount of space used. Recommendations and best practices help you evaluate your environment and get the most out of the online functionality. Finally, we conducted a detailed performance study of online index operations versus offline operations. The details of this study are in [Appendix A](#_Appendix_A:_Performance).

Note   Before you read this paper, we recommend that you read [How Online Index Operations Work](http://msdn2.microsoft.com/en-us/library/ms191261.aspx), [CREATE INDEX (Transact-SQL)](http://msdn2.microsoft.com/en-us/library/ms188783.aspx), and [Lock Modes](http://msdn2.microsoft.com/en-us/library/ms175519.aspx) in SQL Server 2005 Books Online.

#### Data Structures and Concepts

The following data structures and concepts are used by the online index algorithm:

* RID: Row Identifier is used as the locator value in heaps. The RID is built from the file identifier (ID), page number, and number of the row on the page. When a nonclustered index is created on a heap, the index contains the keys of the index and the RID.
* DML: Data Manipulation Language. For the purposes of this paper, this term refers to any operation that alters data in a relational table or index. Examples of DML are insert, delete, and update operations. When select operations apply to this term, they are specifically noted.
* Partitioning: A new feature in SQL Server 2005 Enterprise Edition that allows data to be horizontally partitioned across filegroups by using a specified partitioning scheme. Indexes may be also be partitioned and can be broadly categorized in two areas:
* An *aligned index* uses the same partition scheme and partitioning column as the partitioned table.
* A *nonaligned index* does not use the same partition scheme and partitioning column.

For more information, see [Partitioned Tables and Indexes](http://msdn2.microsoft.com/en-us/library/ms188706.aspx) in SQL Server 2005 Books Online.

#### Overview of the Online Index Build Algorithm

To get the most out of the online index feature, it is important to understand some of the internal processes involved in the online operation. Typically, operations that create, drop, or rebuild indexes acquire and hold exclusive locks on the table or indexed view and prevent any access to the underlying data for the duration of the operation. For example, while a clustered index is being rebuilt by one user, that user and others cannot update or even query the underlying data.

However, when these index operations are performed online, users can perform DML operations against the underlying data for almost the complete duration of the index operation. There are only very short periods of time in which concurrent DML and select operations are prevented.

The online index build process can be roughly divided into three phases:

* Preparation
* Build
* Final

Of these three phases, the build phase is the dominant phase because it is during this phase that the index is rebuilt, dropped, or created. Thus, its duration depends on the size of the data. While the build phase is the longest phase, locks that prevent concurrent DML operations are not held for the duration of this phase. Changes made by concurrent DML operations during the build phase are applied directly to the new index while it is in the process of being built.

The duration of the preparation and final phases is usually short and is independent of the size of the data. The table or indexed view is available to a lesser degree during these phases, depending on the type of index that is being built. These phases and the types of locks used are described in detail in this paper.

The following illustration shows the steps taken during the three phases of online index creation. Note that the steps in each phase are shown in the order in which they occur, but the illustration does not reflect the duration of the steps or the duration of the phases.

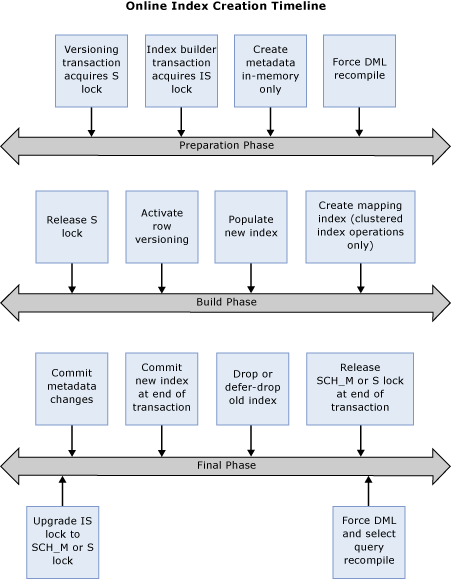


Figure 1

Let’s dive deeper into the three phases and examine the steps in each phase in detail.

##### Preparation Phase

The primary purpose of the preparation phase is to create the index metadata and recompile all DML plans to take the new index into account. In addition, a snapshot of the table is defined; that is, row versioning is used to provide transaction-level read consistency. The preparation phase is roughly divided into the following actions.

1. Locks are acquired.

A versioning transaction acquires a data-share (S) lock on the table or indexed view for the duration of this phase. The reason for the (S) lock is to quiesce updates to the table, thus enabling a stable data snapshot as well as establishing a transition point after which all DML plans must be maintained in the new index. The versioning transaction is committed at the beginning of the build phase as soon as a data snapshot of the source is taken. By data snapshot, we mean row versioning is used to provide transaction-level read consistency. This isolates the index operation from the effects of modifications that are made by other transactions and removes the need for requesting share locks on rows that have been read. Concurrent user update and delete operations during online index operations require space for version records in tempdb. We provide guidelines for estimating this space requirement and an example in [Determining the Amount of Temporary Space Used](#_Determining_the_Amount) later in this paper.

A second transaction, the main index builder transaction, acquires an intent-share (IS) lock on the table. This lock is held until the final phase. The IS lock conflicts with the exclusive (X) lock at the table level and the bulk update (BU) lock, disallowing operations such as the deletion of all rows, BCP operations, or table truncation. The lock is upgraded above an IS mode at specific points as required to synchronize with concurrent DML operations.

Additionally, the main index builder transaction acquires a lock on a special index-operation lock resource to synchronize with concurrent index operations on the table. This lock prevents other concurrent index operations on the same table or indexed view that might cause a conflict in the final phase from starting. For example, a CREATE INDEX operation will conflict with an ALTER INDEX REBUILD operation in the final phase. The index-operation lock resource is visible in the sys.dm\_tran\_locks dynamic management view as resource\_type = ‘OBJECT’ and resource\_subtype = ‘INDEX\_OPERATION’.

1. Metadata is created.

The logical metadata describing the new index, that is, the index name, ID, keys, and attributes, is created and is held only in memory in the metadata cache until the final phase. This is done to avoid violating ID and name uniqueness constraints in the system catalog. For example, when an index is rebuilt online, two versions of the same index with the same name and ID exist during the rebuild operation; the original index and the new, "in-build" index. The new index remains in an "in-build" state until the final phase of the online index operation. From the user’s point of view, the new index is not visible in the **sys.indexes** catalog view until the index operation has completed.

The physical metadata of the index, that is, the definition of the B-tree structure and allocation units, is persisted in the system catalog in this phase and is visible via the **sys.partitions** and **sys.allocation\_units** catalog views.

1. DML plans are recompiled.

After the index metadata is created, an internal-only, per-object, online-version number is incremented before making this new index visible to concurrent connections. The version number change causes all cached DML plans for that table or indexed view to recompile so that the new index, as well as the original index, are maintained going forward. Remember that an (S) lock was acquired at the beginning of the preparation phase. While this lock allows read operations, write operations are blocked, so no DML queries are actually running at this point. Select operations are not affected by the version change because these queries access the original index and simply ignore the "in-build" index.

At this point, everything is set up for the build phase to begin.

##### Build Phase

In short, the build phase is all about populating the new index with sorted data from the existing data source while allowing select and DML operations to continue. Thus, the build phase begins by activating row versioning and releasing the S lock acquired in the preparation phase. Select operations access only the original data source and any preexisting indexes. Concurrent insert, update, and delete operations are applied to both the original data source, any preexisting indexes, and any new indexes being built.

During the build phase the new, empty index structure is populated by scanning and sorting the source data and bulk inserting the data into the new index. Numerous execution plans are possible, depending on the index operation being performed and the data distribution statistics. The query optimizer chooses the plan that is best suited to the type of index that is being created or rebuilt. Note that during a rebuild operation of a clustered or nonclustered index, the existing index is used for the base scan, making a sort operation unnecessary. This saves disk space and CPU resources.

To populate the new index structure, the index builder transaction inserts rows in internal batch transactions. The name of the online batch transaction is "OnlineIndexInsertTxn" and its status can be seen by examining system views such as sys.dm\_tran\_active\_transactions. The duration of each batch transaction is determined dynamically and is generally short-lived. All rows inserted during a specific batch transaction are exclusively locked until the transaction commits and a new batch transaction is started. This means that concurrent DML statements that attempt to touch a row in an active batch transaction are blocked until the batch transaction commits. Deadlocks between the index builder transaction that is holding the batch transaction locks and DML statements are possible, but are handled internally so that neither the DML operation nor the index builder transaction should terminate during the build phase due to a deadlock.

During the build phase, rows in the new "in-build" index may be in an intermediate state called *antimatter*. This mechanism allows concurrent DELETE statements to leave a trace for the index builder transaction to avoid inserting deleted rows. At the end of the index build operation all antimatter rows should be cleared. If an error occurs and antimatter rows remain in the index, DBCC CHECKTABLE and DBCC CHECKDB reports the existence of antimatter rows in errors 5228 and 5229. Rebuilding the index will remove the antimatter rows and resolve the error. For more information, see [MSSQLSERVER\_5229](http://msdn2.microsoft.com/en-us/library/ms365215.aspx) in SQL Server 2005 Books Online.

The algorithm used during the build phase is based on the No Side File (NSF) approach as described in the paper Mohan, C., Narang, I. “[Algorithms for Creating Indexes for Very Large Tables Without Quiescing Updates](http://www.acm.org/sigmod/dl/SIGMOD92/P361.PDF)”, [Proc. ACM SIGMOD International Conference on Management of Data](http://www.informatik.uni-trier.de/~ley/db/conf/sigmod/sigmod92.html), San Diego, June 1992.

###### Mapping Index and Clustered Index Operations

When a clustered index is created, dropped, or rebuilt online, a temporary, nonclustered index, called a *mapping index*, is created in the build phase to map old bookmarks to new bookmarks. A bookmark is simply the row identifier. For example, when creating a clustered index from a heap, the old bookmark is the row ID (RID) of the heap and the new bookmark is the clustering key of the index. The mapping index is used by concurrent DML transactions to determine which records to delete in the new indexes that are being built when rows in the underlying table are updated or deleted. The mapping index is discarded when the index build process commits.

The mapping index contains one record for each row in the table, and it contains the index key columns along with the old and new record identifiers. The mapping index is not used if the clustered index operation is performed offline, or when the operation is on a nonclustered index. The schema of the mapping index is as follows:

Key columns: [new keys], [old keys], [old uniqueifier], [new uniqueifier] where:

* [new keys] are the index keys of the new index being built.
* [old keys] are the keys of the index being dropped. For a heap, this is the RID.
* [old uniqueifier] and [new uniqueifier] are internal integer columns used to disambiguate duplicate keys for nonunique clustered indexes. Thus, these exist only for nonunique indexes where uniqueness is not guaranteed by the specified index keys.

From this schema the disk space requirements for the mapping index can be estimated. We provide guidelines for estimating this space requirement along with an example in [Determining the Amount of Temporary Space Used](#_Determining_the_Amount).

###### Impact of Nonclustered Indexes

If the index operation creates or drops a clustered index and there are nonclustered indexes on the table, the preparation and build phases are repeated twice; once for the new clustered index (or, for a drop operation, the heap), and again for the nonclustered indexes, which are all rebuilt in a single operation to incorporate the change in the definition of the base structure (clustered index or heap). That is, each nonclustered index row must be updated to include either the clustered index key or a row ID (RID) pointing to a data row in the heap. For more information about the structure of nonclustered indexes, see [Nonclustered Index Structures](http://msdn2.microsoft.com/en-us/library/ms177484.aspx) in SQL Server 2005 Books Online.

##### Final Phase

The last phase of the online index operation is to declare to the system that the new index is "ready" and also to clean up any previous states. This phase is roughly divided into the following actions:

1. If the database is mirrored, the index builder transaction waits for the pending log to be shipped and applied to the mirror before proceeding with the lock upgrade defined in step 2. This is done simply to reduce the amount of time the table is not online.
2. The IS lock on the table is upgraded to either a schema-modify (SCH\_M) or data-share (S) depending on the type of online index operation. An S lock is required only when a new nonclustered index is created. If an SCH\_M lock is obtained, conflicting index operations are prevented from starting by using the special index-operation lock resource as described in the [Preparation Phase](#_Preparation_Phase) section.
3. If a clustered index is created, all nonclustered indexes are rebuilt in a single operation and the previous heap is dropped. Similarly, if an index is rebuilt instead of created new, the old version of the index is dropped. Apart from locking the table, the new index itself is also locked in SCH\_M mode during this phase. The index lock resource is visible in the sys.dm\_tran\_locks dynamic management view as resource\_type = 'METADATA' and resource\_subtype = 'INDEXSTATS'. This index lock prevents the new index from being used by others until the index builder transaction is committed. All these locks in the final phase are acquired in the index builder transaction and are held until the end of the transaction. Hence, the table is not available during the final phase until the index builder transaction is committed and locks are released.
4. The logical metadata of the new index, which has been held in memory in the metadata cache up until this point, is persisted in the system catalog. This must be done after the indexes that have been replaced are dropped to avoid violating uniqueness constraints in the system catalog on the index name and ID. The new index is also made "ready." Up until this point, the new index is in an "in-build" state. Declaring the index ready is just an internal state transition on the metadata of the index indicating that it is completely built and usable. If the index was created as part of a PRIMARY or UNIQUE constraint (via ALTER TABLE), the constraint object is created in the catalog.
5. The modified\_date of the table or indexed view is updated to reflect the completion of the index build operation. This is visible in the modified\_date column in the sys.objects catalog view. The modified\_date also acts internally as the schema-version of the object.
6. Updating the schema-version causes another recompile of all cached DML plans that access the table or indexed view. Unlike the recompile operation in the preparation phase, this recompile also includes select queries.
7. The SCH\_M or S lock is released.

#### Capacity Planning and Concurrency Considerations for Index Create and Rebuild Operations

Whether you're performing index create or rebuild operations online or offline, you need to plan for sufficient disk space to accommodate the temporary storage required by these operations. This section provides capacity planning guidelines for estimating the temporary space requirements along with concurrency and contiguity considerations.

##### Temporary Storage Requirements

SQL Server uses temporary storage while creating or rebuilding an index for sorting and other intermediate tasks. Whenever possible, SQL Server uses memory for this work; however, disk space is used when there is insufficient memory. The amount of memory requested for an index operation is controlled by the [index create memory](http://msdn2.microsoft.com/en-us/library/ms175123.aspx) option. For large index operations, it is common that memory will not be sufficient, and disk space will be used.

When disk space is required for temporary storage, it can be used from the space allocated to the user database, or it can be used from the space allocated to the tempdb database. By default, space allocated to the user database is used. However, when the SORT\_IN\_TEMPDB option of the CREATE INDEX or ALTER INDEX statement is set to ON, the tempdb database is used for temporary storage space. The amount of temporary storage space required is the same, regardless of its location. The advantages of using the temporary space from the tempdb database are described in [tempdb and Index Creation](http://msdn2.microsoft.com/en-us/library/ms188281.aspx) in SQL Server 2005 Books Online.

As discussed previously in the [Build Phase](#_Build_Phase) section, row versioning is used for online index create and rebuild operations. If there are concurrent DML transactions on the table during the online index create or rebuild, row versions are maintained in the version store. The version store is always created in the **tempdb** database. For more information about row versioning, see [Row Versioning-based Isolation Levels in the Database Engine](http://msdn2.microsoft.com/en-us/library/ms177404.aspx) and [Row Versioning Resource Usage](http://msdn2.microsoft.com/en-us/library/ms175492.aspx) in SQL Server 2005 Books Online.

Together, the SORT\_IN\_TEMPDB and ONLINE index options influence the amount of temporary space used in the user database and the tempdb database. Other factors influence this as well, such as whether you are dealing with a clustered or nonclustered index and whether you are creating or rebuilding the index. So, how much temporary space is required? The first step is to know the tools available to measure temporary disk space usage.

##### Measuring Temporary Disk Space Usage

When the temporary space is used from the tempdb database, you can measure the amount of temporary space used by an index operation by using the dynamic management views provided in SQL Server 2005. There are three views that report the temporary disk space used by any operation in tempdb:

* sys.dm\_db\_task\_space\_usage
* sys.dm\_db\_session\_space\_usage
* sys.dm\_db\_file\_space\_usage

While these views only pertain to the tempdb database, you can set the SORT\_IN\_TEMPDB option to ON when testing for disk space usage requirements and then plan for the same space allocation in your user database.

The sys.dm\_db\_task\_space\_usage dynamic management view provides **tempdb** usage information for each task. As a task (such as an index rebuild) progresses, you can monitor how much temporary space the task is using. However, as soon as the task completes, the counters in the view are reset to zero. So, unless you happen to query this view just at the moment before the task completes, you can’t get the total amount of **tempdb** space used by a given task. However, when the task is completed these values are aggregated at the session level and stored in the sys.dm\_db\_session\_space\_usage view.

The sys.dm\_db\_session\_space\_usage provides tempdb usage information for each session. The easiest way to measure the **tempdb** space used by a given operation is to query sys.dm\_db\_session\_space\_usage for your session before and after the operation. However, there is a catch. The data in sys.dm\_db\_session\_space\_usage is not updated until the completion of the batch; therefore, you must execute these statements as three separate batches. Essentially, all you really need is three GO statements, as shown in the following example:

SELECT \* FROM sys.dm\_db\_session\_space\_usage WHERE session\_id = @@spid;

GO

<create or rebuild index statement>

GO

SELECT \* FROM sys.dm\_db\_session\_space\_usage WHERE session\_id = @@spid;

GO

When you query the sys.dm\_db\_session\_space\_usage view, pay attention to the following two columns in the result set:

* internal\_objects\_alloc\_page\_count: This column represents the space used by the sort runs while creating or rebuilding an index.
* user\_objects\_alloc\_page\_count: This column represents the **tempdb** space used by the temporary mapping index. The temporary mapping index is created only when an online index operation creates, drops, or rebuilds a clustered index.

To measure the size of the version store, you can query the version\_store\_reserved\_page\_count column in the sys.dm\_db\_file\_space\_usage view. The version store size can also be monitored by using the System Monitor (perfmon) counter Version Store Size (KB) in the Transactions performance object. The amount of space required for the version store depends on the size and duration of the transactions that change the data in the underlying table.

For more information about these dynamic management views, see [sys.dm\_db\_session\_space\_usage](http://msdn2.microsoft.com/en-us/library/ms187938.aspx), [sys.dm\_db\_task\_space\_usage](http://msdn2.microsoft.com/en-us/library/ms190288.aspx), and [sys.dm\_db\_file\_space\_usage](http://msdn2.microsoft.com/en-us/library/ms174412.aspx) in SQL Server 2005 Books Online.

For information about monitoring the query execution and performance of online index operations, see [Appendix B](#_Appendix_B:_Diagnostics).

##### Determining the Amount of Temporary Space Used

As discussed earlier, the amount of temporary space used depends upon various factors. The following list summarizes the important items to note:

* For offline create or rebuild operations, **tempdb** is used only for sorting when SORT\_IN\_TEMPDB is ON. By default this option is set to OFF and the sort takes place in the space allocated to the user database.
* For online create or rebuild operations of a clustered index, additional temporary space is used for the mapping index.
* When you rebuild a clustered index, sort runs are not needed because the data is already sorted. Therefore, no temporary space is required. However, when you create a clustered index, sorting is required, and temporary space is used for the sort runs.
* Temporary space for sorting is not required when you rebuild a clustered index offline because the data is already sorted. In addition, the offline operation does not create a temporary mapping index.
* Temporary space for sorting is not required when you rebuild a nonclustered index online because the data is already sorted.
* Partitioning and index uniqueness do not affect the amount of temporary space used.

The following table summarizes when and how space is used for index create and rebuild operations in tempdb or in a user database.

Table 1   S = sort runs, M = mapping index, V = version store

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation is Online | SORT\_IN\_  TEMPDB is On | Temporary Space for Clustered Index Create | Temporary Space for Clustered Index Rebuild | Temporary Space for Nonclustered Index Create | Temporary Space for Nonclustered Index Rebuild |
| Yes | Yes | S, M, V in tempdb | M, V in tempdb  S not required | S, V in tempdb  M, not required | V in tempdb |
| Yes | No | S and M in user database  V in tempdb | M in user database  V in tempdb  S not required | S in user database  V in tempdb  M not required | V in tempdb |
| No | Yes | S in tempdb  M, V not required | None required | S in tempdb  M, V not required | S in tempdb  M, V not required |
| No | No | S in user database  M, V not required | None required | S in user database  M, V not required | S in user database  M, V not required |

Table 2 illustrates the amount of temporary space used with respect to the size of the index for indexes of different width. That is, it shows the ratio of temporary space to the size of the index (temporary space used / index size). For the purposes of this example, we assume that the total row size is 46 bytes and the indexes are defined as follows:

* A single-column index on an IDENTITY column. The indexed column is 4 bytes, or approximately 8% of the row size. This is an example of a narrow index.
* A composite index of three columns. The total size of the indexed columns is 10 bytes (total 46 bytes in the row), or approximately 25% of the row size. This is an example of a typical index.
* A composite index comprising all columns. The total size of the indexed columns is 46 bytes or 100% of the row size. This is an example of the widest possible index. Such an index is not very common on large tables, but is used here as an example of the worst-case scenario.

The space required for the version store is not included in this table.

Table 2   Ratio of temporary space to the size of the index

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index Operation | Ratio of temporary space to the size of the index | | | | | | | | |
| Single-column index  8% of row size  (4 bytes out of 46 bytes) | | | Three-column index  25% of row size  (10 bytes out of 46 bytes) | | | All-column index  100% of row size  (46 bytes out of 46 bytes) | | |
| mapping index | sort runs | total | mapping index | sort runs | total | mapping index | sort runs | total |
| Online clustered index create | 0.33 | 1.1 | 1.43 | 0.52 | 1.12 | 1.64 | 1.21 | 1.11 | 2.32 |
| Online clustered index rebuild | 0.17 | 0 | 0.17 | 0.31 | 0 | 0.31 | 1.01 | 0 | 1.01 |
| Online nonclustered index create | 0 | 1.07 | 1.07 | 0 | 1.05 | 1.05 | 0 | 1.02 | 1.02 |
| Online nonclustered index rebuild | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Offline clustered index create | 0 | 1.02 | 1.02 | 0 | 1.02 | 1.02 | 0 | 1.01 | 1.01 |
| Offline clustered index rebuild | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Offline nonclustered index create or rebuild | 0 | 1.07 | 1.07 | 0 | 1.05 | 1.05 | 0 | 1.02 | 1.02 |

###### Example

To further illustrate the temporary space requirements shown in Table 2, the following example extrapolates the values for a three-column index (10 bytes) shown above for a specified table. For this example, we will create a clustered index from a heap with the SORT\_IN\_TEMPDB option set to ON and ONLINE set to ON. The heap and resulting clustered index have the following size characteristics:

* Table size: 1,159,633 pages (8.85 GB)
* Number of rows in the table: 179,743,073
* Row size: 46 bytes
* Number of columns in the index: 3 (10 bytes total)
* Size of the clustered index created in the user database: 1,159,650 pages (8.85 GB)

Given that information, we can use the values in Table 2 to determine the approximate temporary space required to create the clustered index.

* The size of the temporary mapping index in tempdb will be approximately 52% of the index size or 603,848 pages (4.6 GB).
* The space used in tempdb for sort runs will be approximately 112% of the index size, or 1,293,728 pages (9.88 GB).
* Total temporary space required is approximately 164% of the index size, or 1,897,576 pages (14.48 GB).

Therefore, for the duration of the online index operation, you need at least 8.85 GB of free space in the user database to store the clustered index and 14.48 GB of free space in tempdb. After the clustered index is created, the space used by the table is freed up, but during the creation of the clustered index, the table and the index coexist so you need free space in the user database equal to the size of the index being created. If SORT\_IN\_TEMPDB is set to OFF, you need a total of 23.33 GB of temporary space (14.48 + 8.85 GB) in the user database.

In addition, if there are concurrent transactions on the table during the index operation, additional space in tempdb is required for the version store. In one test, we updated 5% of the rows in the table while creating the clustered index online, and the version store was measured to be 143,408 pages, or 1.1 GB for this operation.

###### Observations

Based on our tests, the following observations were made:

* The space used for sort runs and the mapping index, with respect to the index size, is almost constant for the size of the table. We verified these results with various table sizes starting with 1x to 8x where 1x = 179 million rows. The ratios illustrated in Table 2 are consistent across the table sizes.
* The space used for sort runs, with respect to the index size, is almost constant for clustered or nonclustered indexes, whether it is a create or a rebuild operation, whether it is performed online or offline, and whether it is a wide or a narrow index. The required space for sort runs is approximately 110% of the size of the index.
* The size of the mapping index increases as the index gets wider. For the worst-case scenario, that is when the clustered index comprises all the columns in the table, the mapping index is approximately 121% of the size of the index.
* If the table already has a clustered index and a new clustered index is created on a different set of columns by dropping the old index (DROP\_EXISTING = ON), the size of the mapping index is dependent upon the old clustered index (source index) as well. The size of the mapping index increases as the source index gets wider. See the [Build Phase](#_Build_Phase) section for a description of the mapping index schema.
* The mapping index is smaller for a rebuild operation than for a create operation. This is because the row identifiers don’t change in a rebuild operation; the new and old identifiers are the same. Therefore, only one row identifier is included in the mapping index.

#### Online Index Best Practices

In the previous section we discussed the temporary disk space needed for most index operations and the tools you can use to estimate disk space requirements for these operations. In this section, we recommend the following ways to improve index operation efficiency and optimize your environment for index operations:

* Determine the index operation needed for your environment.
* Use ONLINE and SORT\_IN\_TEMPDB options effectively.
* Manage transaction log space.
* Control parallelism and memory resources.

In SQL Server 2005, some limitations apply to the indexes that can use the online feature. For example, index operations cannot be performed online for XML indexes and indexes on local temporary tables. For a complete list of limitations, see [Appendix D](#_Appendix_D:_Online).

##### Determine the Index Operation Needed for Your Environment

To determine which index operation is required and whether it should be performed online or offline, evaluate your environment and specific requirements. It may be optimal to run index operations offline. While this means users cannot access the data during the operation, the operation finishes faster and uses fewer system resources. However, some production environments require 24-hour-a-day, 7-day-a-week user access to the data, making routine index maintenance operations difficult or even impossible to schedule. In these environments, running such operations online can be an effective option. The most frequent online index operations used by the DBA in a such an environment are:

* Defragmenting indexes

Heavily fragmented indexes can degrade query performance and cause your application to respond slowly. Index fragmentation is removed by rebuilding or reorganizing the index. Depending on the size of the data, these operations may take a few minutes or several hours. The method of reorganizing an index, ALTER INDEX REORGANIZE, is always an online operation and uses minimal resources. However, reorganizing an index is not as effective as rebuilding and should not be used on heavily fragmented indexes. The method for rebuilding an index, ALTER INDEX REBUILD, drops the index and creates a new one. In doing this, fragmentation is removed and disk space is reclaimed by compacting the pages. Rebuilding an index can be executed online or offline. For more information about these methods, see [Reorganizing and Rebuilding Indexes](http://msdn2.microsoft.com/en-us/library/ms189858.aspx) in SQL Server 2005 Books Online and the [Index Defragmentation Best Practices](http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/ss2kidbp.mspx) white paper.

* Creating ad-hoc nonclustered indexes

While troubleshooting and resolving query performance, you may need to create nonclustered indexes on the fly. Creating these indexes online ensures that users are not denied access to the underlying data.

##### Use ONLINE and SORT\_IN\_TEMPDB Effectively

The ONLINE and SORT\_IN\_TEMPDB index options affect both the temporary space requirements and performance of the index create or rebuild operation. The advantages and disadvantages of each are covered in this section.

When considering the ONLINE option, you must weigh the need for a performant index operation versus the need for concurrent user access to the underlying data.

* To achieve the best performance, that is, the least time to create or rebuild an index, set ONLINE to OFF. However, this prevents all user access to the underlying table for the duration of the index create or rebuild operation.
* To achieve the best concurrency, that is, the least impact on other users accessing the table, set ONLINE to ON. However, the index operation will take more time.

You must also take into consideration the extra temporary space requirements of the online operation.

* To use the least amount temporary space while rebuilding a clustered index, set ONLINE to OFF.
* To use the least amount of temporary space while rebuilding a nonclustered index, set ONLINE to ON.
* If there are concurrent user transactions on the table during the online index operation, you must plan for additional space in tempdb for the version store.

For more information, see [Determining the Amount of Temporary Space Used](#_Determining_the_Amount) in this paper.

As we discussed earlier, when SORT\_IN\_TEMPDB is set to ON, sort runs and other intermediate tasks are stored in tempdb rather than the user database. Setting this option to ON can have two advantages:

* You can achieve the most contiguous space in the index. When the sort extents are held separately in tempdb, the sequence in which they are freed has no affect on the location of the index extents. Also, when the intermediate sort runs are stored in tempdb instead of the destination filegroup, there is more space available in the destination filegroup. This increases the chance that index extents will be contiguous. For more information, see [tempdb and Index Creation](http://msdn2.microsoft.com/en-us/library/ms188281.aspx) in SQL Server 2005 Books Online.
* When both SORT\_IN\_TEMPDB and ONLINE are set to ON, the index transactions are stored in the tempdb transaction log, and the concurrent user transactions are stored in the transaction log of the user database. This allows you to truncate the transaction log of the user database during the index operation if needed. Additionally, if the tempdb log is not on the same disk as the user database log, the two logs are not competing for the same disk space.

###### Maximize tempdb Space and Throughput

If you use SORT\_IN\_TEMPDB or ONLINE, it is important that the tempdb database is sized appropriately and has adequate I/O throughput. We recommend the following guidelines:

* Ensure that tempdb is on a disk subsystem that provides sufficient I/O throughput, and the database is big enough to accommodate the temporary space required for the index create or rebuild operation.
* Move tempdb to a storage device with sufficient space and performance. By default, tempdb is created in the data directory under the SQL Server installation folder C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data. In this configuration, there may not be enough space for tempdb and the storage may not have adequate I/O throughput. For information about moving **tempdb**, see [Moving System Databases](http://msdn2.microsoft.com/en-us/library/ms345408.aspx) in SQL Server 2005 Books Online.
* Keep in mind that tempdb is a global resource for all users. When you plan for the temporary space needed in tempdb for the index operation, consider the activities in all user databases that may be using **tempdb**.

For more information about tempdb capacity planning and best practices, see [Working with tempdb in SQL Server 2005](http://www.microsoft.com/technet/prodtechnol/sql/2005/workingwithtempdb.mspx).

##### Manage Transaction Log Space

Large-scale index operations, whether offline or online, can generate large data loads that can cause the transaction log to quickly fill. To make sure that the index operation can be rolled back, the transaction log cannot be truncated until the index operation has been completed. Therefore, the transaction log must have sufficient space to store both the index operation transactions and any concurrent user transactions for the duration of the index operation. Note, however, that the log can be backed up during the index operation.

When you run large-scale index operations, consider the following recommendations:

1. Make sure that the transaction log has been backed up and truncated before running large-scale index operations, and that the log has sufficient space to store the projected index and user transactions.
2. Set the recovery model of the user database to SIMPLE or BULK\_LOGGED before running the index operation. Both of these models allow for minimal logging of the index operation. Minimal logging is more efficient than full logging and reduces the chance of the index operation filling the log space.
3. In an OLTP environment with concurrent DML operations, consider setting the SORT\_IN\_TEMPDB option to ON. This separates the index transactions from the concurrent user transactions. The index transactions will be stored in the tempdb transaction log, and the concurrent user transactions will be stored in the transaction log of the user database. This allows for the transaction log of the user database to be truncated during the index operation if it is required. Additionally, if the tempdb log is not on the same disk as the user database log, the two logs are not competing for the same disk space. The recovery model of tempdb is set to SIMPLE and cannot be changed.
4. Do not run the online index operation in an explicit transaction. The log will not be truncated until the explicit transaction ends.

###### Transaction Log Test Measurements

We tested five index operations on various index types to measure the transaction log space generated under each recovery model. The following table shows the results of these tests. For test details, see [Appendix A](#_Appendix_A:_Performance). You can use these values to estimate the transaction log space required for your index operations. For an example of how to measure transaction log space used for a particular online index operation, see [Appendix C](#_Appendix_C:_Log).

Table 3   Transaction log test measurements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test Description | Log-to-Data Ratio using FULL | Log-to-Data Ratio using BULK\_LOGGED | Log-to-Data Ratio using SIMPLE |
| 1 | Create a nonclustered online index | 1.15 | 0.14 | 0.13 |
| 2 | Rebuild a clustered index | 1.34 | 0.14 | 0.14 |
| 3 | Rebuild a nonclustered index when the clustered index exists | 1.14 | 0.16 | 0.16 |
| 4 | Aligned partitioned index: Create a nonclustered index on a partitioned clustered index using the same partition scheme | 1.16 | 0.15 | 0.15 |
| 5 | Nonaligned partitioned index: Create a nonclustered index on a partitioned clustered index using a different partition scheme | 1.16 | 0.15 | 0.15 |

##### Control Parallelism and Memory Resources

On a large, multiprocessor system, the build phase may use more processors to perform the operations associated with the index operation. While this improves the performance of the index operation, doing so may cause insufficient memory and worker thread resources for other applications and database operations for the duration of the index operation. Thus, for certain server configurations such as 32-bit systems where virtual address space is at a premium, or other systems with high memory usage, it may be advisable to use the MAXDOP index option or the max degree of parallelism (DOP) option in sp\_configure to control the number of processors dedicated to the index operation. In this way, you can balance the resources that are used by the index operation with those of the concurrent users. Note that the MAXDOP index option applies only to the index operation in which it is specified, whereas, changing the max degree of parallelism (DOP) option affects all queries in the instance of SQL Server.

###### Parallelism and Partitioned Indexes

Both aligned and nonaligned parallel partitioned index plans build each partition serially. Thus, when creating or rebuilding a partitioned index over *n* partitions, the maximum degree of parallelism is the smaller of *n* partitions or the number of CPUs. Parallelism offers the most benefit when the rows are equally balanced across partitions because the work is load balanced.

Nonaligned index build plans generally have higher disk space requirements than aligned partitioned indexes because all the partitions are built simultaneously. For aligned index build plans, only a subset of partitions is built at any one moment in time.

For both aligned and nonaligned indexes, the greater the degree of parallelism, the greater the memory requirement. For example, if the degree of parallelism is set to 4, a nonaligned partitioned index with 100 partitions requires at a minimum, sufficient memory for four processors to sort 4,000 pages at the same time.. If the partitioned index is aligned, the memory requirement is reduced to four processors sorting 40 pages, or 160 (4 \* 40) pages. For more information, see [Special Guidelines for Partitioned Indexes](http://msdn2.microsoft.com/en-us/library/ms187526.aspx) in SQL Server 2005 Books Online.

#### Conclusion

This paper describes the online index functionality available in SQL Server 2005 Enterprise Edition. The performance and disk space usage analysis provides guidelines on how much capacity to plan for when running index operations online. The paper provides an overview of the supportability mechanisms available that allow the database administrator to view the current status of an online operation. Lastly, specific recommendations and best practices are provided that will help you avoid common pitfalls and errors that we have seen in the field.

#### Appendix A: Performance Study

We conducted a detailed study on the performance aspect of online index operations versus offline operations. The study is divided into two categories:

* Absolute index build performance when there is no other concurrent activity on the server
* Index build performance when there is a concurrent workload on the affected table

The study shows that an offline index operation is, in general, faster than an equivalent online index operation, but the operations scale in a similar fashion.

We also compared the performance of online index operations over an increasing number of partitions. The study shows that the number of partitions does not significantly affect the scaling of the online index operation.

##### Test Cases and Set Up Details

We used representative OLTP workloads (an order management system that tracks orders, payments, deliveries, and related information) in all of our tests. The tests used a high volume of random I/O, very low CPU usage, and a large number of concurrent users workloads. To measure the impact on system performance, the following combinations of workloads were used:

* OLTP workloads without concurrent online index operations
* Online index operations without concurrent workloads
* Online Index operations with concurrent OLTP workloads

Clustered and nonclustered indexes were built on the most intensively updated columns.

In addition to measuring performance impact, we also measured log space requirements for online index operations. The results of these measurements are discussed in [Manage Transaction Log Space](#_Manage_Transaction_Log).

Database properties

The test database had the following database size and file properties.

Database size: 63.48 GB

Table 4   Database file specifications

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Filegroup | Size | Maxsize | Growth | Usage |
| MSSQL\_oltp\_root | PRIMARY | 8 MB | Unlimited | 10% | data |
| MSSQL\_oltp\_log | NULL | 48.82 GB | Unlimited | 10% | log |
| MSSQL\_oltp\_e | MSSQL\_oltp\_fg | 14.65 GB | Unlimited | 10% | data |

Data files were located on the same physical disk; log files were located on a different physical disk.

Test machine configuration

System:

* Microsoft Windows Server™ 2003 Enterprise Edition SP1
* Microsoft SQL Server 2005 Enterprise Edition SP1

Computer:

* Intel(R) XEON(TM)MP CPU
* 2.00GHz
* 1.99GHz, 3.87GB of RAM

##### Performance Without Concurrent Activity

We conducted three tests in this category:

* Creating a nonpartitioned, unique, clustered index on a heap
* Creating a nonpartitioned, nonclustered index from a clustered index
* Creating a partitioned clustered index from a heap

Because the performance of rebuild operations is similar to create operations, they are not listed separately for these tests.

We conducted the tests on servers with 1, 2, 4, and 8 CPUs (degree of parallelism). For the nonpartitioned index creation operations, we compared the performance of the online index operation against the same index operation performed offline. For the partitioned index test, we compared the performance of the index operation over a number of partitions ranging from 0 to 200.

###### Test1: Creating a Nonpartitioned Unique Clustered Index on a Heap

The following illustration shows the response times of offline and online index operations that create a nonpartitioned, unique, clustered index from a heap. In this test, an integer column was used as the index key.

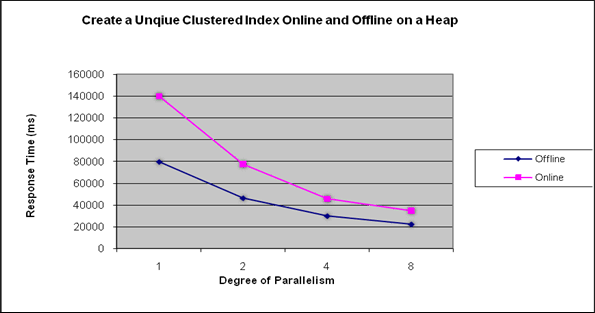


Figure 2

Here are the comparative results between the offline and online operations.

|  |  |
| --- | --- |
| Degree of Parallelism | Ratio Online/Offline |
| 1 | 1.75 |
| 2 | 1.67 |
| 4 | 1.52 |
| 8 | 1.55 |

|  |  |  |
| --- | --- | --- |
|  | Scale Factor | |
| Degree of Parallelism | Offline | Online |
| 1 | 1 | 1 |
| 2 | 1.72 | 1.80 |
| 4 | 1.54 | 1.69 |
| 8 | 1.34 | 1.31 |

As we see from the data, the online clustered index operation ran 1.5-1.7 times slower than the offline operation, but it scales up in a similar fashion. The slowness of the online index operation is due to the maintenance of the temporary mapping index during the build phase.

###### Test 2: Creating a Nonpartitioned Nonclustered Index on a Clustered Index

The following illustration shows the response times of offline and online index operations that create a nonpartitioned, nonclustered index on a clustered index. In this test, an integer column was used as the index key.

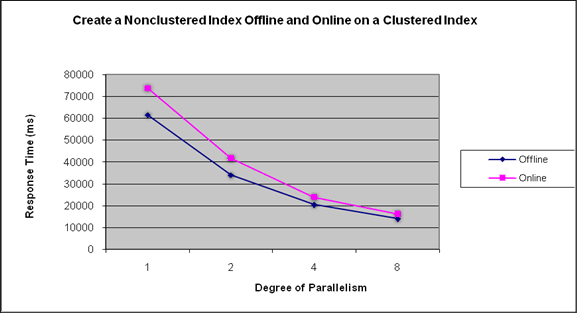


Figure 3

Here are the comparative results between the offline and online operations.

|  |  |
| --- | --- |
| Degree of Parallelism | Ratio Online/Offline |
| 1 | 1.19 |
| 2 | 1.22 |
| 4 | 1.17 |
| 8 | 1.16 |

|  |  |  |
| --- | --- | --- |
|  | Scale Factor | |
| Degree of Parallelism | Offline | Online |
| 1 | 1 | 1 |
| 2 | 1.81 | 1.77 |
| 4 | 1.66 | 1.74 |
| 8 | 1.45 | 1.47 |

The online nonclustered index operation is 1.1 to 1.2 times slower than the offline operation. Because the online nonclustered index operation does not require the extra temporary mapping index, its performance is closer to that of the offline operation.

###### Test 3: Creating a Partitioned Clustered Index

The following illustration shows the response times of online index operations that create a partitioned, clustered index from a heap. In this test, an integer column was used as the index key and the number of partitions range from 0 to 200.

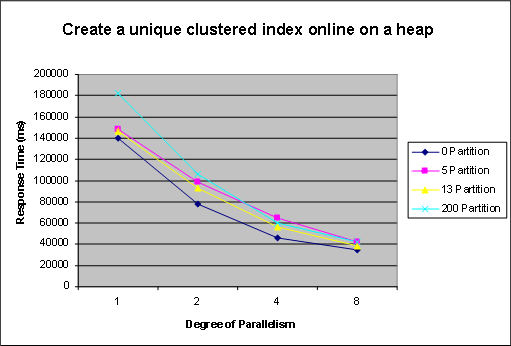


Figure 4

Here are the comparative results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Degree of Parallelism | Response Time per Number of Partitions (ms) | | | |
|  | 0 | 5 | 13 | 200 |
| 1 | 140370 | 148623 | 146936 | 181826 |
| 2 | 77780 | 98513 | 93280 | 106533 |
| 4 | 45843 | 64860 | 56500 | 60493 |
| 8 | 34876 | 42720 | 38140 | 42156 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scale Factor | | | | |
| Degree of Parallelism | Number of Partitions | | | |
|  | 0 | 5 | 13 | 200 |
| 2 | 1.80 | 1.50 | 1.57 | 1.70 |
| 4 | 1.69 | 1.51 | 1.65 | 1.76 |
| 8 | 1.31 | 1.51 | 1.48 | 1.43 |

As we see, the number of partitions does not significantly affect the scaling of the online index operation.

##### Performance with Concurrent Activities

In this test category we analyzed both the performance impact of the concurrent workload on the online index operation and the impact of the online index operation on the concurrent workload using a highly intensive OLTP workload. All the tests were conducted on a table containing 15,000,000 rows.

###### Concurrent Workload Impact on Online Index Operation

The following table shows the impact of a highly intensive, concurrent workload on an online index build operation. The run time numbers are shown in milliseconds (ms) with the baseline (no concurrent activity) being 10 ms.

|  |  |  |
| --- | --- | --- |
| Test Number | Test Description | Total Run Time: Online Index Operation + Concurrent Workload (ms) |
| 1 | Create a nonclustered index on an existing clustered index. | 44.5 |
| 2 | Rebuild a clustered index. | 591 |
| 3 | Rebuild a nonclustered index. | 44.5 |
| 4 | Create an aligned nonclustered index on a partitioned clustered index. | 123.5 |
| 5 | Create a nonaligned nonclustered index on a partitioned clustered index. | 104.2 |

Overall, the performance of the online index create operation dropped significantly under heavy concurrent activity, usually around 5-15 times. This is expected, as the heavy OLTP workload consumed most of the CPU resources during the tests.

We observed a significant slowdown in the execution of the clustered index rebuild operation (Test 2) compared to the other tests. This is mainly due to the intensive concurrent workload and maintenance of the temporary mapping index. In addition, the larger size of the clustered index compared to the nonclustered index resulted in more data and log copying/movement.

###### Online Index Operation Impact on Concurrent Workload

The following table shows the impact an online index operation has on a concurrent workload. The data is measured in TPMC, which roughly translates to throughput. Thus, a smaller number means less work was done.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TPMC (baseline 100% without an online index operation) | Workload + Test1 (%) | Workload + Test 2 (%) | Workload + Test 3 (%) | Workload + Test 4 (%) | Workload + Test 5(%) |
| Weighted average value | 32.24 | 47.58 | 59.95 | 71.87 | 79.73 |

Test 1, creating a nonclustered index, affected the concurrent workload more than Test 3, rebuilding a nonclustered index. This is mainly due to the extra sort required to create the index. The memory consumption from the sort affected the workload significantly because the concurrent OLTP workload used was very sensitive to the amount of memory available.

##### Test Case Summary

Based on our tests, we can make the following observations:

* In general, online index operations are always slower than offline operations.

This is due to the additional steps that are required to support the online process. For example, concurrent insert, update, and delete operations must be applied to the original data source, any preexisting indexes, and any new indexes being built. In addition, a temporary mapping index is created and maintained during clustered index operations.

Without taking concurrent user activity into account, online nonclustered index operations were approximately 1.1 to 1.2 times slower than equivalent offline operations. Online clustered index operations are approximately 1.5 to 1.7 times slower than equivalent offline operations.

* The primary factors that impact the performance of online index operations are:
* The level of concurrent user activity.
* The size of the underlying data.
* Maintenance of the temporary mapping index. This applies only to clustered index operations.
* A higher degree of parallelism can improve the performance of online index operations. However, with high concurrent activity, it is better to balance the CPU resources that are used by the index operation with those of the concurrent users so that sufficient memory and worker thread resources are available to support the concurrent activity.
* The number of partitions has minimal impact on the scaling of the online index operation.

#### Appendix B: Diagnostics

To monitor query execution and performance information of online index operations, use the following Transact-SQL statements and trace events.

* SET STATISTICS XML ON / SET STATISTICS PROFILE ON
* These statements return information after the statement executes. The output includes the numbers of rows scanned. Extra information, such as Memory Grant, is included in the XML output.
* These statements return the row distribution for parallel plans. If the distribution is skewed, creating or updating statistics on the table prior to the index operation new statistics on the table may help correct the problem.
* SET STATISTICS IO ON to return the number of logical/physical reads from the disk.
* SET STATISTICS TIME ON to monitor CPU time.
* Use the Progress Report: Online Index Operation event class to monitor the progress of an online index operation while the operation is running.

#### Appendix C: Log Measurement Script

The following script demonstrates how to measure the transaction log usage of a particular online index operation. In this example, a create index operation is used. To obtain accurate results, this script should be executed without concurrent activity on the database used.

DECLARE @initiallog int;

DECLARE @finallog int;

DECLARE @differenceinlog int;

DECLARE @datasize int;

DECLARE @logtodataratio decimal(19,10);

--Find initial log size:

SELECT @initiallog = cntr\_value FROM master..sysperfinfo

WHERE instance\_name = '<database\_name>' AND counter\_name = 'Log File(s) Used Size (KB)';

--Create an Index:

CREATE UNIQUE CLUSTERED INDEX <index name> ON <table\_name> (<column\_list>)

WITH (ONLINE = ON);

--Find new log size:

SELECT @finallog = cntr\_value FROM master..sysperfinfo

WHERE instance\_name = '<database\_name>' AND counter\_name = 'Log File(s) Used Size (KB)';

--Find change in log size:

SET @differenceinlog = @finallog - @initiallog;

--find index size:

DECLARE @dbid smallint, @objid int;

SET @dbid = DB\_ID('<database\_name>');

SET @objid = OBJECT\_ID('<table\_name>');

SELECT @datasize = SUM(page\_count) FROM sys.dm\_db\_index\_physical\_stats(@dbid,@objid,null,null,null)

WHERE index\_type\_desc = 'CLUSTERED INDEX'; --In case of nonclustered index - 'NONCLUSTERED INDEX'

SET @datasize = @datasize \* 8; -- #\_of\_pages\*8

--Find Log-to-Data ratio

SET @logtodataratio = CAST(@differenceinlog AS decimal(19,10))/CAST(@datasize AS decimal(19,10));

SELECT 'Initial Log'= @initiallog, 'Final Log'=@finallog,

'Difference in Log'=@differenceinlog, 'Data Size'=@datasize,

'Log-to-data ratio'=@logtodataratio;

GO

#### Appendix D: Online Index Limitations

The limitations common to all online operations are noted in the following bulleted points. The exception conditions are noted in the following table.

* Any online operation on XML index
* Any online operation on local temp table
* LOB columns being part of INCLUDE column clause
* Clustered index operations with LOB columns being part of a base table
* Unique nonclustered index creation, either via CREATE INDEX or ALTER TABLE ADD CONSTRAINT. This limitation does not apply to SQL Server 2005 SP1 and later versions.

|  |  |  |
| --- | --- | --- |
| Online index operation | Excluded indexes | Other restrictions |
| ALTER INDEX REBUILD | Disabled clustered index or disabled indexed view  XML index | Specifying the keyword ALL may cause the operation to fail when the table contains an excluded index.  Single partition rebuild.  Additional restrictions on rebuilding disabled indexes apply. For more information, see [Guidelines for Disabling Indexes](ms-help://MS.SQLCC.v9/MS.SQLSVR.v9.en/udb9/html/2198f1af-fa44-47e9-92df-f4fde322ba18.htm) in SQL Server 2005 Books Online. |
| CREATE INDEX | XML index  Initial unique clustered index on a view |  |
| CREATE INDEX WITH DROP\_EXISTING | Disabled clustered index or disabled indexed view | Converting clustered index to nonclustered index and vice-versa. |
| DROP INDEX | Disabled indexes  XML index  Nonclustered indexes | Multiple indexes cannot be specified within a single statement. |
| ALTER TABLE ADD CONSTRAINT (PRIMARY KEY or UNIQUE)  Or  ALTER TABLE DROP CONSTRAINT (PRIMARY KEY or UNIQUE with CLUSTERED INDEX option) |  | Only one subclause is allowed at a time. For example, you cannot add and drop PRIMARY KEY or UNIQUE constraints in the same ALTER TABLE statement. |

For more information:

<http://www.microsoft.com/technet/prodtechnol/sql/default.mspx>

Did this paper help you? Please give us your feedback. On a scale of 1 (poor) to 5 (excellent), [how would you rate this paper](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20Online%20Index%20Operations)?