Kofax TotalAgility

Database Guidance

Version: 1.0

05 December 2018

Table of Contents

Introduction

Database configuration planning and structure includes, but is not limited to:

- RDMS platform
- On/Off premise
- Disk platform
- AIO or horizontally scaled Kofax TotalAgility databases
- High Availability / Load Balancing
- Backup schedule
- Disaster recovery

This document provides guidance on how to manage your Kofax TotalAgility databases once you have already made configuration / deployment choices and does not contradict or supersede any best practice guidance from Microsoft®. Use this document to assist in making informed decisions regarding SQL Server configuration and performance improvements for your Kofax TotalAgility system.

Distributed versus non-distributed transactions

TransactionScope is an important class in the .NET Framework. The main responsibility of this class is supporting transactions from a code block. Kofax TotalAgility uses this class for managing both **local** and **distributed transactions** from TotalAgility code.

TransactionScope uses the LTM - Lightweight Transaction Manager in .NET. If you open more than one connection in the same transaction or go between databases, TransactionScope will automatically promote the transaction to the two-phase commit transaction manager, the Microsoft Distributed Transaction Coordinator (MSDTC).

Note: You must use MSDTC as required when deploying the Kofax TotalAgility databases you choose to separate into individual databases, such as Audit, Archive, Live, and Documents.

Microsoft Distributed Transaction Coordinator

Microsoft Distributed Transaction Coordinator (MSDTC) is a Windows service component used to coordinate the transaction infrastructure for distributed systems such as file systems, message queues,

and databases. In this case, a transaction generally structures interactions between autonomous agents in a distributed system.

Each transaction is a state transformation with four key properties commonly referred to as the ACID properties:

1. Atomic (all or nothing)
2. Consistent (legal)
3. Isolated (independent of concurrent transactions)
4. Durable (once it happens, it cannot be reversed)

There are different techniques that implement the ACID properties but the most commonly used is the two-phase commit.

Configuration differences

Configuration differences exist for MSDTC, depending on its deployment. For example:

- On a standalone server, MSDTC runs as a normal service.

  On a failover cluster, MSDTC has to be installed as a cluster resource. With Windows Server 2003 or earlier, MSDTC can be assigned to the cluster group and use the quorum disk. With Windows Server 2008 and later, MSDTC must have its own group and disk.

  On a SQL Server 2012 availability group cluster, where SQL Server is installed as a standalone instance on each node, MSDTC has to run as a normal service.

  If you have multiple instances of SQL Server installed on a standalone server or cluster, you must have only one instance of MSDTC per server or cluster.

MSDTC default timeouts

The default timeout for MSDTC is 60 seconds. A value of 0 indicates NO TIMEOUT. **Use this setting with extreme caution**. The setting means the transaction will NEVER timeout. Long-running processes in the database will continue consuming resources until the transaction is completed or aborted (via other environmental factors).

It is worth noting that often (but not always) an MSDTC timeout is indicative of insufficient resources in relation to SQL configuration and or network bandwidth/latency.

By default, all the distributed transactions initiated using System.Transactions have the default timeout of 10 minutes. This value can be updated in the machine.config file to set the maxTimeout to another value, such as 1 hour:

```
<configuration>
 <system.transactions>
  <machineSettings maxTimeout="01:00:00" />
 </system.transactions>
</configuration>
```

You may also have to modify the System.Transactions section in machine.config by setting the **allowExeDefinition** attribute to "MachineToApplication" (from the default MachineOnly) depending on how you have the MSDTC deployed. For example:

```
<sectionGroup name="system.transactions"
type="System.Transactions.Configuration.TransactionsSectionGroup,
System.Transactions, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, Custom=null">
      <section name="defaultSettings"
type="System.Transactions.Configuration.DefaultSettingsSection,
System.Transactions, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, Custom=null" />
      <section name="machineSettings"
type="System.Transactions.Configuration.MachineSettingsSection,
System.Transactions, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, Custom=null"
allowDefinition="MachineOnly" allowExeDefinition="MachineToApplication"
/>
    </sectionGroup>
```

The valid values for this attribute are:

- **MachineOnly:** The ConfigurationSection can be defined only in the Machine.config file.
- **MachineToApplication:** The ConfigurationSection can be defined either in the Machine.config file or in the Exe.config file in the client application directory. This is the default value.
- **MachineToLocalUser:** The ConfigurationSection can be defined in the Machine.config **MachineToLocalUser** file, the Exe.config file in the client application directory, the User.config file in the roaming user directory, or the User.config file in the local user directory.
- **MachineToRoamingUser:** The ConfigurationSection can be defined in the Machine.config file, the Exe.config file in the client application directory, or the User.config file in the roaming user directory.

Database space planning

At first glance, database space planning seems straightforward and simple. In reality, many different factors must be considered, and space planning is different for each customer.

Although the Professional Services / PreSales and Product organizations will offer assistance on database sizing, a fairly simplistic approach is to estimate the future size of the database based on:

1. The row length of each table.
2. The number of business transactions that the database will need to store over a given period of time

This approach may work fairly well for databases with a relatively small number of tables. However, for Kofax TotalAgility, this does not take into account the variable length columns in the tables or the size of the documents being stored.

You also need to plan for the amount of space required by indexes, as the indexes can occupy the same, or even more, space than the table data. It is also very important to factor in that indexes and data do not fully utilize the index (data) pages. The amount of free space in data pages depends on the number of columns in a table, and the data type of each column. Not utilizing a full 64 KB page can lead to increased space requirements due to varying degrees of fragmentation.

Application features that retain data must be factored in, and some features may not be obvious.

1. Job history
2. Variable history
3. Auditing
4. Process design loop iterations
5. Synchronous processing

Note that process design loop iterations are often overlooked. Keep in mind that some processes have iterations that exceed tens of thousands, which can have a significant impact on database storage.

Transaction Logs are vitally important

Every application-related update, delete and insert statement will cause the transaction log to grow.

Even in Simple Recovery mode, the transaction log is still used. The only difference is it automatically reclaims log space to keep space requirements small, essentially eliminating the need to manage the transaction log space.

In practice this means every transaction is still written to the transaction log, but once the transaction is complete and the data is written to the data file, the space used in the transaction log file is now re-usable by new transactions (it doesn't require a checkpoint or a shrink).

Customers should be aware that a Database Version upgrade for Kofax TotalAgility can involve significant constraint and table changes. Prior to an upgrade, it is very important to analyze the current database size and plan accordingly for potential growth.

This query will list the total table size inclusive of clustered index, heap and all non-clustered indices:

```
SELECT
   s.Name AS SchemaName,
   t.NAME AS TableName,
   p.rows AS RowCounts,
   SUM(a.total_pages) * 8 AS TotalSpaceKB,
   SUM(a.used_pages) * 8 AS UsedSpaceKB,
   (SUM(a.total_pages) - SUM(a.used_pages)) * 8 AS UnusedSpaceKB

FROM
   sys.tables t
INNER JOIN
   sys.schemas s ON s.schema_id = t.schema_id
INNER JOIN
   sys.indexes i ON t.OBJECT_ID = i.object_id
INNER JOIN
   sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
INNER JOIN
   sys.allocation_units a ON p.partition_id = a.container_id
WHERE
   t.NAME NOT LIKE 'dt%'    -- filter out system tables for diagramming
   AND t.is_ms_shipped = 0
   AND i.OBJECT_ID > 255
GROUP BY
   t.Name, s.Name, p.Rows
ORDER BY
   s.Name, t.Name
```

To separate a table space from index space, you need to use AND i.index_id IN (0,1) for the table space (index_id = 0 is the heap space, index_id = 1 is the size of the clustered index = data pages) and AND i.index_id > 1 for the index-only space.

As a general rule of thumb, if you assume an increase of 50% for the largest table affected by the upgrade, you can allocate for sufficient log growth.

Using Queuing Theory

The Queueing Theory rule stipulates that when resource utilization reaches a certain point (the knee of the curve on a utilization graph), the queue of processes waiting for the resources and the response time of the system start to increase exponentially (negatively).

For disk space, and the number of I/O operations per second, this ''knee'' occurs between 80-85% capacities. Thus a database (or collection thereof) should not occupy more than 85% of total disk space; and the number of I/O's per second should not exceed 85% of its maximum.

TempDB

The TempDB system database, a **global resource** available to all users connected to the instance of SQL Server, is used to hold the following:

1. Temporary user objects that are explicitly created, such as global or local temporary tables, temporary stored procedures, table variables, or cursors.
2. Internal objects that are created by the SQL Server Database Engine, such as work tables to store intermediate results for spools or sorting.
3. Row versions that are generated by data modification transactions in a database that uses read-committed using row versioning isolation or snapshot isolation transactions.
4. Row versions that are generated by data modification transactions for features, such as online index operations, Multiple Active Result Sets (MARS), and AFTER triggers.

Don't be fooled by its name, TempDB. Do not treat "temporary" as any less important than your application databases. While most DBAs remember to place TempDB on a fast disk drive (RAID) separate from data files, they do allocate enough space to adequately manage the complete server interactions.

If TempDB is not adequately sized, it will increase the total number of I/O operations on the drive where TempDB resides and slow down database performance.

Rebuilding the database

When a database is frequently updated via INSERT, UPDATE, or DELETE statements, we can expect it to become fragmented over time.

If database indexes are heavily fragmented, the SQL Server query optimiser may choose a non-optimal execution plan when using an index to resolve a query.

To gauge the level of fragmentation, you can use this simple query that will list every index on every table in your database, ordered by percentage of index fragmentation.

```
SELECT dbschemas.[name] as 'Schema',
dbtables.[name] as 'Table',
dbindexes.[name] as 'Index',
indexstats.avg_fragmentation_in_percent,
indexstats.page_count
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL, NULL, NULL) AS indexstats
INNER JOIN sys.tables dbtables on dbtables.[object_id] = indexstats.[object_id]
INNER JOIN sys.schemas dbschemas on dbtables.[schema_id] = dbschemas.[schema_id]
INNER JOIN sys.indexes AS dbindexes ON dbindexes.[object_id] = indexstats.[object_id]
AND indexstats.index_id = dbindexes.index_id
WHERE indexstats.database_id = DB_ID()
ORDER BY indexstats.avg_fragmentation_in_percent desc
```

There is no hard and fast rule that states when you should rebuild an index, but conservatively an index rebuild should not be attempted until fragmentation is between 60%-80%. HOWEVER, there will be a point (which is different for each customer) whereby rebuilding the indices will have no material impact as the underlying table and index storage cannot be defragmented into full 64 KB pages.

Index rebuild modes of operation

SQL gives you two options regarding index rebuilds.

1. ONLINE. The new index is built while the old index is accessible to reads and writes. Any update on the old index is also applied to the new index. An antimatter column is used to track possible conflicts between the updates and the rebuild (such as deletion of a row that was not yet copied). When the process is completed, the table is locked for a brief period and the new index replaces the old index. If the index contains LOB columns, ONLINE operations are not supported in SQL Server 2005/2008 R2.

2. OFFLINE. The table is locked up front for any read or write, and then the new index is built from the old index, while holding a lock on the table. No read or write operation is permitted on the table while the index is being rebuilt. Only when the operation is done is the lock on the table released so that reads and writes are allowed again.

Online index operations require more disk space requirements than offline index operations.

- During index creation and index rebuild operations, additional space is required for the index being built (or rebuilt).
- In addition, disk space is required for the temporary mapping index. This temporary index is used in online index operations that create, rebuild, or drop a clustered index.
- Dropping a clustered index online requires as much space as creating (or rebuilding) a clustered index online.

You must weigh the time and resources required to do the index rebuild (along with the associated impact on the application) against the potential gain in storage and query optimisation. Rebuilding an index that will fragment to the same or similar levels afterward is an exercise in futility.

Parallelism

You can use a parallel plan for a database system to improve input/output processing speed during operations such as index building, data loading, and query execution.

The decision to use a parallel plan to execute the query or not depends on multiple factors. At a minimum, SQL Server should be installed on a multi-processor server, the requested number of threads should be available to be satisfied, the Maximum Degree of Parallelism option is not set to 1 and the cost of the query exceeds the previously configured Cost Threshold for Parallelism value.

See the Microsoft parallelism guidelines: https://support.microsoft.com/en-gb/help/2806535/recommendations-and-guidelines-for-the-max-degree-of-parallelism-confi

1. If you have very small number of queries executing at the same time compared with the number of processors, you can set the MAXDOP value to a larger value. For example, you can set the MAXDOP value to 16.
2. If you a have very large number of queries executing at the same time compared with the number of processors, you can set the MAXDOP value to a smaller value. For example, you can set the MAXDOP value to 4.

Setting MAXDOP = 1 will in effect prevent any form of parallelism from occurring.

CXPACKET wait and parallelism

CXPACKET waits are merely a symptom of either a consumer or provider waiting on some other resources, such as IO, CPU, or memory grant. Their presence is indicative of issues with Parallelism.

Inefficiencies related to query parallelism can be found by running the **sys.dm_os_wait_stats** view. The following script will report on the current wait times and percentages for each wait type.

```
WITH Waits AS
(
SELECT
  wait_type,
  wait_time_ms / 1000. AS wait_time_s,
  100. * wait_time_ms / SUM(wait_time_ms) OVER() AS pct,
  ROW_NUMBER() OVER(ORDER BY wait_time_ms DESC) AS rn
FROM sys.dm_os_wait_stats
WHERE wait_type
  NOT IN
```

```
    ('CLR_SEMAPHORE', 'LAZYWRITER_SLEEP', 'RESOURCE_QUEUE',
  'SLEEP_TASK', 'SLEEP_SYSTEMTASK', 'SQLTRACE_BUFFER_FLUSH', 'WAITFOR',
  'CLR_AUTO_EVENT', 'CLR_MANUAL_EVENT')
  ) -- filter out additional irrelevant waits

SELECT W1.wait_type,
 CAST(W1.wait_time_s AS DECIMAL(12, 2)) AS wait_time_s,
 CAST(W1.pct AS DECIMAL(12, 2)) AS pct,
 CAST(SUM(W2.pct) AS DECIMAL(12, 2)) AS running_pct
FROM Waits AS W1
 INNER JOIN Waits AS W2 ON W2.rn <= W1.rn
GROUP BY W1.rn,
 W1.wait_type,
 W1.wait_time_s,
 W1.pct
HAVING SUM(W2.pct) - W1.pct < 95; -- percentage threshold;
```

It is also misleading to jump to the conclusion that all CXPACKET waits are always a bad thing. From SQL 2016 and later, the “bad” wait, where work amongst threads is distributed unevenly, is reported as a CXPACKET wait. However, the “good” type of wait, where the consumer threads are just waiting for all the producer threads to do their work, is reported under a new wait called CXCONSUMER. Prior to that, ‘’good’’ or “bad’ are reported under CXPACKET.

Cost Threshold for Parallelism

The cost threshold for the parallelism (CTFP) option specifies the threshold at which SQL Server creates and runs parallel plans for queries. SQL Server creates and runs a parallel plan for a query only when the estimated cost to run a serial plan for the same query is higher than the value set in **cost threshold for parallelism**. The cost refers to an estimated cost required to run the serial plan on a specific hardware configuration.

CTFP can be set to any value from 0 through 32767. The default value is 5. For many applications, including Kofax TotalAgility, this is too low. A suggested minimum is 35 but each environment is different and the CTFP value should be adjusted to prevent unwanted parallelism. Microsoft guidelines suggest a value between 25-50.

Unwanted Parallelism

You may ask if there is such a thing as unwanted parallelism. The short answer is yes. Kofax TotalAgility is not explicitly designed to utilize parallelism. So in a deployment whereby MAXDOP is greater than 1 and the default CTFP has been reached, the query optimizer will invoke parallelism. If you have long-

running queries or locking issues, this has a negative impact by increasing the SQL resources resulting in exponentially reducing performance.

However, in certain cases, a parallel plan may be chosen even though the query's cost plan is less than the current cost threshold for parallelism value. This can happen because the decision to use a parallel or serial plan is based on a cost estimate provided before the full optimization is complete.

Important Microsoft Message

In relation to MAXDOP and CTFP (discussed above), note the following:

“Any value that you consider using should be thoroughly tested against the specific application activity or pattern of queries before you implement that value on a production server.”

Normal performance

Customers often report issues with SQL performance and typically report that “it’s gotten slower.” When you ask them to quantify what that means, they often struggle to relate the difference in a measurable, quantitative manner. If you don't know what the performance characteristics are when your system is running normally, how can you know which performance characteristics are abnormal?

Baseline, Benchmark, Watermark

Baseline and benchmark are similar but distinct activities. Think along the lines of:

- A Baseline being a value against which later measurements and performance can be compared. The baseline could represent a minimum set of users and system load.
- A Benchmark is a test of higher levels of a predefined load (such as a sanity or regression test) for comparison against a baseline. For example, a benchmark may compare the number of jobs processed to completion in a given time period.
    - A High Watermark is a special type of benchmarking. It establishes the maximum load at which the server can operate without performance degradation.

The purpose of baselining and benchmarking is that you know, definitively, what is normal and abnormal for your SQL system. The baseline is about identification of a significant state, meaning your set of numbers meet an approval status from the LOB. The benchmark is about assessing the relative performance.

SQL Server Health

Baseline metrics literally define what is normal for the database application's performance. The following should be looked at as a minimum list but offers a great insight into SQL Server health.

Operating system Indicators

- **Processor(_Total)\% Processor Time:** How busy are the server's CPUs? A basic indicator to help us know that a server is running well within acceptable operating parameters. Normally we expect to see this counter in the 20 to 40 percent range.
- **Memory\Available Mbytes:** Is server memory an issue? Are there other processes on the server besides SQL Server? Are other processes taking memory SQL Server needs?
- **Paging File(_Total)\% Usage:** If the OS starts to run out of memory, it takes large chunks of that memory and swaps it out to disk, to the Paging File. Regardless of the physical media involved, writing to disk is the slowest operation. This counter will show if you are encountering memory issues.
- **PhysicalDisk(_Total)\Avg. Disk sec/Read & PhysicalDisk(_Total)\Avg. Disk sec/Write:** How fast is the I/O subsystem responding to requests for data from the operating system (latency)? Latency values should normally not be more than 20ms; if you're using SSD, probably not more than 5ms. If you see latency values of a second or more, your I/O subsystem has issues.
- **System\Processor Queue Length:** The number of threads that are waiting for time on the system processor. If greater than 0, there are more requests per core than the system can manage.
- **Network interface\Bytes total/sec:** The rate at which the network adapter is processing data. This number should remain lower than 60% of the maximum sustainable bandwidth of the network.
- **SQL Server performance counters**.
- **SQLServer:Access Methods\Forwarded Records/sec:** How fragmented your heaps are. Each time a search for data encounters a forwarding pointer, it increments this counter.
- **SQLServer:Access Methods\Page Splits/sec:** How fragmented your tables are. If SQL needs to insert a row onto a page, and there isn't room, SQL Server will split the page into multiple pages, move rows from one page to another to balance the pages out, and then insert the row. This is a very expensive and time-consuming process.
- **SQLServer:Buffer Manager\Buffer cache hit ratio:** The percentage of times a needed page was already in the buffer pool when it was required.
- **SQLServer:Buffer Manager\Page life expectancy:** What process is causing the pages in the buffer pool to be flushed out.

- **SQLServer:General Statistics\Processes blocked:** Blocked processes are inevitable and SQL Server has mechanisms to handle them. However, if the value exceeds the normal range (for your system) you'll want to identify excessive blocking due to page escalation.
- **SQLServer:SQL Statistics\Batch Requests/sec:** How busy is your SQL Server?
- **SQLServer:SQL Statistics\SQL Compilations/sec & SQLServer:SQL Statistics\SQL Re-Compilations/sec:** How often does SQL Server compile or recompile query plans because the plan in the cache is no longer valid, or there's no plan in the cache for this query?

Note: With a default instance, the first part of the counter name (to left of colon) is SQLServer. With a named instance, such as KTA01, it's MSSQL$ followed by the instance name, such as MSSQL$KTA01.