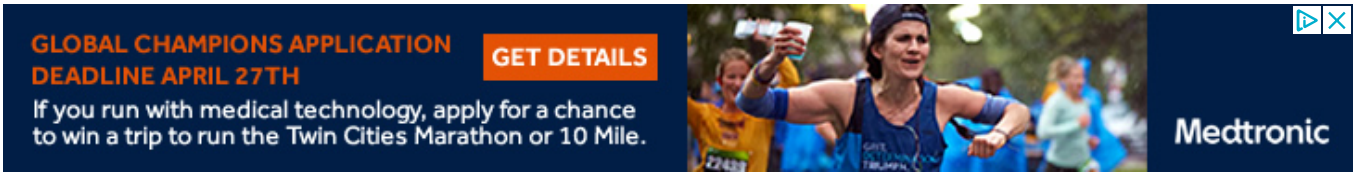
MSSQLTips Free SQL Server Conference Registration + Travel Expenses click to learn more

Minimize performance impact of SQL Server DBCC CHECKDB



Free Webcast - Simplify SQL Server Administration

Problem

Every company wants to ensure the database integrity of their database systems. If you are not running a regular database consistency check against your databases, then your databases could have corruption without your knowledge. Therefore, it is important to run DBCC CHECKDB as part of your daily, weekly, monthly, etc. database maintenance routine. In fact, in my previous article on this topic, I showed you how we can setup a process to Capture and Store SQL Server Database Integrity History using DBCC CHECKDB. However, there is a cost to running DBCC CHECKDB as well. Clearly, the pros outweigh the cons, but there are considerations. What is the performance overhead of running a DBCC CHECKDB on your production systems, as it is unavoidably an IO intensive operation? I have a need to run DBCC CHECKDB without any performance implications. Do you have any suggestions? Read this tip to learn more.

Solution

The purpose of this tip is to educate the community on the following:

- How the DBCC CHECKDB works
- The potential performance impact
- Considerations when running the command
- Possible workarounds or solutions to
  - Ensure there is no database corruption
  - How to minimize the impact executing DBCC CHECKDB on a 24x7 production environment

DBCC CHECKDB Overview

In general, DBCC CHECKDB should be executed while the database is ONLINE, but it is recommended that there is minimal activity on the SQL Server during this operation. It does not acquire table locks by default. Instead, it acquires schema locks that prevent metadata changes, but allow data changes. The schema locks acquired will prevent the user from getting an exclusive table lock required to build a clustered index, drop an index, truncate the table, etc.

With the latest version of DBCC CHECKDB, it performs CHECKALLOC, CHECKTABLE and CHECKCATALOG with the one command rather than running the commands separately. These commands run in a linear fashion. So, it is a very thorough database consistency check, looking at page allocation on disk, checking and ensuring that the page structures are properly linked, and finally checks the system tables. Here is a simple sample command:

```
DBCC CHECKDB (SQLCentric);
GO
```

Assuming the results from the command above are successful, and you see a similar message that states "**CHECKDB found 0 allocation errors and 0 consistency errors in database 'YourDB'**", you can be confident that the database is consistent, has no corruption, and properly stored on your disk.

Execute DBCC CHECKDB on a Database Snapshot

As of SQL Server 2005, all the DBCC validation commands use **database snapshots**. Therefore, this is considered on ONLINE operation, and will keep validation operations from interfering with on-going database operations. Using snapshots allow the validation operation to see a consistent view of the data. A snapshot is created at the beginning of the CHECK command, and no locks are acquired on any of the objects being checked. The actual check operation is performed against the snapshot. Unlike regular snapshots, the "snapshot file" that DBCC CHECKDB creates cannot be configured and is invisible to the end user. This in itself, does not minimize IO, but the advantages here are to avoid any locking contention on the production database. The caveat is that it always uses space on the same disk volume as the database being checked, and therefore you must ensure adequate space to perform this operation.

You can avoid creating a snapshot and save disk space by using the WITH TABLOCK option of the DBCC command. This essentially can and should be only be done OFFLINE. Using the WITH TABLOCK option will

force the DBCC to try to grab an exclusive database lock which will prevent users from accessing the database if you attempt this command during production hours.

```
DBCC CHECKDB (SQLCentric) WITH TABLOCK; -- This option does NOT use a snapshot
GO
```

You can see from the results of using WITH TABLOCK that a snapshot cannot use or check certain consistencies, as per the message below:

***"DBCC* CHECKDB** will not check SQL Server catalog or Service Broker consistency because a database snapshot could not be created or because WITH TABLOCK was specified." Source - http://support.microsoft.com/kb/928518.

Also, keep in mind, during regular maintenance, DBCC CHECKDB should NOT be used with any of the repair options, as the database is put into single-user mode, so no other transactions can be altering data. In this case too, a snapshot is not created.

As a point of reference, here are the Repair options available with DBCC CHECKDB:

- REPAIR_ALLOW_DATA_LOSS
  - Tries to repair all reported errors. These repairs can cause some data loss.
- REPAIR_FAST
  - Maintains syntax for backward compatibility only. No repair actions are performed.
- REPAIR_REBUILD
  - Performs repairs that have no possibility of data loss. This can include quick repairs, such as repairing missing rows in non-clustered indexes, and more time-consuming repairs, such as rebuilding an index.

For example, if you wish to use DBCC Repair, you must first put the database in single user mode as shown below:

```
ALTER DATABASE SQLCentric
SET single_user WITH ROLLBACK IMMEDIATE;
GO
DBCC CHECKDB (SQLCentric, repair_allow_data_loss);
GO
```

TempDB and IO Intensity

Just by the nature of the command, DBCC CHECKDB is an IO intensive operation. Above, we learned by using an internal snapshot, we can avoid potential locking and blocking. However, another concern is related to IO caused by spooling to the **TempDB** database. Because the DBCC CHECKDB process makes very heavy use of the TempDB, Microsoft suggests storing TempDB on its own separate set of spindles. With properly configured disks, throughput will be much better.

Duration and Performance Impact of DBCC CHECKDB ONLINE

If the decision is to run DBCC CHECKDB online, then in addition to the above strategies, I would recommend recording and measuring the performance impact and duration. First, test the duration of the DBCC CHECKDB against a restored copy of the database(s), so you can record an base duration. Next, schedule the process to run in a maintenance window where there is minimal user activity. You may already have statistics on peak vs. non-peak hours, and can schedule the code to run, at the lowest level of activity. Based on this information you should have a good sense of the time needed.

DBCC CHECKDB WITH PHYSICAL_ONLY

One alternative to minimize contention, is to use the WITH PHYSICAL_ONLY DBCC CHECKDB option. This option limits the processing to checking the integrity of the physical structure of the page and record headers in addition to the consistency between the pages for the allocation structures (data and indexes). This option is specifically designed to provide a low overhead check of the physical consistency of the database. This check also detects torn pages and common hardware failures that can compromise a user's data. Perhaps you would run this more frequently and then schedule a full DBCC CHECKDB during periods of low activity. The full run of DBCC CHECKDB will take considerably longer to complete, due to the more comprehensive logical structure checks.

Here is the sample command:

```
DBCC CHECKDB (SQLCentric) WITH PHYSICAL_ONLY;
GO
```

Backup, Restore and Run DBCC CHECKDB

Another alternative is to run the DBCC CHECKDB on another SQL Server. You can setup a process where you restore the database to another server and run DBCC CHECKDB against it. Since the backup process is a bit-by-bit copy of the database, upon restoring the database it will be in exactly the same state as your online database. If the database is inconsistent or there are storage or allocation problems, the backup will store these too. As such, the results of any possible corruption should surface on the restore. This will completely eliminate the IO issue on the production server. However, the backup, copy and restore process could be quite lengthy, and will also require the correct amount of disk space to accommodate the database.

Backup WITH CHECKSUM

The Backup WITH CHECKSUM option was introduced with SQL Server 2005 and offers some protection that occurs along with your regular backup routine. Although the "Backup WITH CHECKSUM" option will cause all existing page checksums on allocated pages to be checked, there may be some database corruption that will not be detected. Even after the checksums are enabled and the database has checksum page verification on, not all pages in the database will have page checksums. Furthermore, the I/O subsystem does not distinguish between pages with and without page checksums when causing corruption. Therefore, it is NOT a replacement for DBCC CHECKDB, however, this option will give you some insight into the integrity of your database backup.

In addition, the backup with checksum command does not perform all of the same sets of verification. However, reading and checking all the checksums can identify some corruption. Page checksums are checked during a backup and the backup will fail if any of the page checksums are wrong, which is an indication that there is corruption. With this option, the backups should not take significantly longer, but this option does not eliminate the need for regular DBCC CHECKDB routines.

Below is a simple example of backing up the database with the checksum option:

```
BACKUP DATABASE SQLCentric TO DISK = 'd:\sqldbbackups\sqlcentric.bak' WITH CHECKSUM;
GO
```

DBCC CHECKDB and Database Mirroring

If you have database mirroring setup, you can create a database snapshot explicitly from the mirror and run DBCC CHECKDB on that version of the database. Be aware that the I\O load from this could make the REDO queue on the mirror larger and cause the failover to take longer. This could cause an issue for your SLA. In addition, Microsoft may require a license for this SQL Server, although you are not using the mirror for any data processing.

Furthermore, the mirror would have to actually *hit* the corrupt page as part of REDO for it to get the auto-page-repair - otherwise you would never know there is corruption until the mirroring failover occurs and a CHECKDB happens on the new principal database. For more information, check out this article - "Mirror, Mirror".

With respect to corruption, one great feature in SQL Server 2008 R2 is the ability to automatically repair corrupt pages during database mirroring. See this MSDN article on Automatic Page Repair During a Database Mirroring Session. Automatic page repair is an asynchronous process that runs in the background.

You can actually view stored history on suspect pages by querying the 'suspect_pages' table, introduced in SQL Server 2005:

```
SELECT *
FROM msdb..suspect_pages;
GO
```

You can specifically look for bad pages such as bad checksum, torn page errors by filtering by event type, as such:

```
-- Select nonspecific 824, bad checksum, and torn page errors.
SELECT *
FROM msdb..suspect_pages
WHERE (event_type = 1 OR event_type = 2 OR event_type = 3);
GO
```

Third Party Tools

A final option is using a third party tool where you can "restore" a database from your backup and run DBCC checks against it. You are able to do so while never actually allocating disk space for the backup because it is a virtual database. Using one of these tools effectively off-loads the DBCC CHECKDB from the production system. In general, the advantages of using one of these tools are that you will reduce storage overhead (of having to run DBCC against a large restored database), eliminate the load on the production system and ensure you have a consistent backup.

The reality is that although we seek ways to minimize the performance overhead when running DBCC CHECKDB, there is NO way to run consistency checks on a database without IO impact. Also be aware that running CHECKDB, even on the production database does not give you an absolute guarantee that there is no corruption. It just says that at the time the individual pages were read and processed, they did not have any issues. As soon as the page have been read, the IO subsystem could corrupt one. Therefore, the need for regular consistency checks is essential. Hopefully, this tip will help you think about the importance of running DBCC CHECKDB, and provide some ways to minimize the performance impact to your database systems.

Next Steps

- How often are you running DBCC CHECKDB to make sure your databases are free of corruption? If you have not had a window of opportunity to ensure your databases are free of corruption, check out some of the options in this tip to see what could help your organization.
- Check out these related resources:
  - Want to know more in-depth about DBCC CHECKDB, check out Paul Randal's blog on CHECKDB From Every Angle...
  - Learn about the DBCC CHECKDB with DATA_PURITY command by Greg Robidoux
  - Checkout my previous tip on how to Capture and Store SQL Server Database Integrity History using DBCC CHECKDB.
  - Microsoft MSDN Library on DBCC CHECKDB - with all options described as well as Optimizing DBCC CHECKDB Performance.

Last Update: 2011-06-02

You're invited
to our next free
SQL Server webcast!

click to learn more

About the author

Robert Pearl is a SQL MVP, and President of Pearl Knowledge Solutions, Inc., offering SQLCentric monitoring and DBA services.

View all my tips

Related Resources

- SQL Server Database Integrity Checks Checklist...
- Minimize performance impact of SQL Server DBCC CHE...
- Ensure SQL Server Data Purity Checks are Performed...
- SQL Server 2005 DBCC CHECKDB with DATA_PURITY comm...
- Using DBCC PAGE to Examine SQL Server Table and In...
- More SQL Server DBA Tips...

Post a comment or let the author know this tip helped.

All comments are reviewed, so stay on subject or we may delete your comment. Note: your email address is not published. Required fields are marked with an asterisk (*).

*Name *Email Notify for updates

*** NOTE *** - If you want to include code from SQL Server Management Studio (SSMS) in your post, please copy the code from SSMS and paste the code into a text editor like NotePad before copying the code below to remove the SSMS formatting.