Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Volume 21, No. 1, ISSN 1726-9679
ISBN 978-3-901509-73-5, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2010
Make Harmony Between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals of DAAAM International

PROBLEM WITH RAPID TRANSACTION LOG GROWTH IN SQL SERVER DATABASES

KORBAR, D[amir]

Abstract: *Database files in SQL Server can be set to grow automatically when they need more space. However, if not set up and maintained properly, databases can sometimes expand too much in a short time. Rapid database growth can eventually end with disk being full and thus cause serious problems. In most cases, it is the database transaction log file that grows too big. This paper explains what can cause a rapid transaction log growth, how to shrink it when it has already expanded too much, and more important, how to prevent it from growing too large.*
Key words: *SQL Server, databases, transaction log, recovery models*

1. INTRODUCTION

Every SQL Server database consists of at least one data file and one transaction log. While data file contains actual data, transaction log contains changes that need to be propagated to data files. Database engine fully relies on transaction log when committing or rolling back transactions. It is also essential for ensuring data consistency during the database recovery process.

Data files grow as the result of inserting new data into the database. Updating or deleting data does not enlarge data file. However, that is not the case with transaction log. Since every data change has to be recorded in transaction log, it can also grow while updating or deleting data.

When creating a new database we can specify the initial size of its constituent files and thus reserve a certain amount of free disk space for the database usage. After the database has been working for some time, the newly added data can occupy all the free space. If transaction log becomes full, no further data changes are possible because every change requires a new entry and consequently additional amount of free space in transaction log. In such a situation, transaction log has to expand in order for the database to continue working.

Enlarging database files can be done manually by database administrators or automatically by database engine. By default, SQL Server database files grow automatically (data files by 1 MB and transaction log by ten percent of their size) as soon as they become full. If proper care is not taken, database files can grow too large and, in the worst scenario, use all of the free disk space. If there are really a huge quantity of data that needs to be kept in a database, there is not much to do with data files besides from getting a big enough disk. However, transaction log can still be kept on a reasonable size. This is because database engine does not need all the records from the transaction log all the time. After they are not needed for the case of database recovery any more, the space they occupy can be reused.

Under default settings, reusing of unneeded transaction log space does not occur automatically and it requires some database maintenance tasks to be done. Sometimes organizations or firms lack database administrators (Mullins, 2002) so that other technical stuff, like application developers

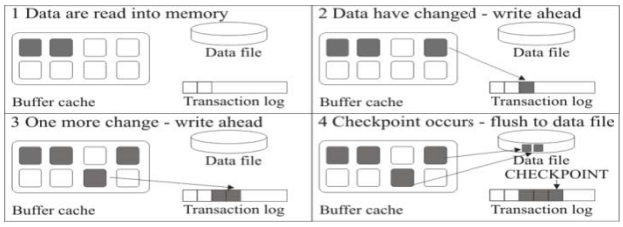

Fig. 1. Propagating data changes from memory to disk

or system administrators, set up databases. However, since they are not so well trained in working with databases, they may not apply proper database administration techniques, which could generally be bad for the overall database performance and availability. Shrinking transaction log file and preventing it from growing too big is one of the database administration issues that requires good understanding and proper database maintenance.

2. UNDERSTANDING TRANSACTION LOG

SQL Server transaction log is a write-ahead log, which means that every data change must be written to transaction log before it is recorded to data file. By using a write-ahead log, SQL Server maintains the ACID properties for a transaction (Microsoft, 2009). In order to save a data change to the database, SQL Server firstly has to make that change in buffer cache in main memory and only after that, the change is propagated to disk. The change is not automatically saved to the data file, but rather to the transaction log. This means that the existence of data pages in main memory that do not hold the same state as their corresponding data pages in data files is possible. Such pages are called dirty pages. SQL Server periodically triggers a process called checkpoint. When checkpoint occurs, all dirty pages are written from buffer cache to data files. This process of updating the contents of data pages on disk is called flushing. Figure 1 illustrates the process of propagating data changes from memory to disk.

Records are written to transaction log file sequentially, one after another. Figure 2 shows how a transaction log could look like at the moment of database failure. Transaction Tran 1 had been committed before the last checkpoint occurred so all of its changes were flushed to disk during that checkpoint. Tran 2 was committed only after the last checkpoint so data affected by Tran 2 are still inconsistent in data file. To recover the database and to put it back in a consistent state, SQL Server needs log records from Tran 2, whereas records belonging to Tran 1 are not needed any more. Unnecessary records written before Tran 2 are coloured gray.

SQL Server can put a special mark on transaction log records that are not needed any more in process of database recovery so that later it can overwrite them with new records. Marking unnecessary log records is known as log truncation.

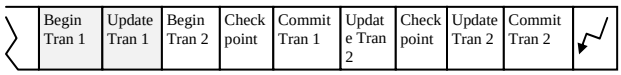

Fig. 2. Transaction log records before database failed

Transaction log is normally truncated during the process of taking a transaction log backup copy.

3. SQL SERVER RECOVERY MODELS

SQL Server can operate in three different modes, called recovery models: simple, bulk logged and full. The main difference between these three models is the level of details that are written to transaction log. Consequently, the level of database recoverability differs among them. In full recovery model, every change is recorded to the transaction log with the most details. This enables database to be recovered to any desired point in past. In bulk logged recovery model, bulk loads and some other operations are not recorded with many details. This gives the possibility of point-in-time recovery unless these minimally logged operations occur. Finally, databases working in simple recovery model record database changes with minimal possible details which still ensure database recoverability in case of its failure. Under the simple recovery model database can be recovered only to a point when last full or differential backup was taken.

If we accept default settings when setting up a new database, it operates in full recovery model and database files are set to grow automatically. It is crucial to note that the only way to truncate transaction log under this recovery model is to make a transaction log backup copy. If a log backup hasn't been done, transaction log will never be truncated and it will always expand when it becomes full. In bulk-logged recovery model, backing up transaction log is also the only way to truncate transaction log, but it is little less likely that transaction log will grow so fast because some operations that produce high number of log records in full recovery model are not logged here. Under the simple recovery model, SQL Server database engine truncates transaction log automatically when checkpoint occurs. Making transaction log backup copies is not possible under this recovery model.

4. SHRINKING TRANSACTION LOG

It is very important to realize that truncating transaction log does not reduce its size. It only marks records that can be reused. Reducing transaction log size is done by log shrinking. When SQL Server shrinks the transaction log file, it in fact releases truncated parts of the log to the operating system and thus transaction log becomes smaller. We cannot shrink transaction log if it has never been truncated before.

When database operates in full recovery model, it means that we cannot shrink transaction log if we have never made its log backup. Let's suppose that we have a database DB1 and we have to shrink its log DB1_log, which has never been backed up. We could execute the following T-SQL statements:

```
Use master; Backup log DB1 to disk = '< backup path >';
Use DB1; DBCC SHRINKFILE (DB1_log, 0);
```

If transaction log occupies all the free disk space and database becomes unavailable, we probably do not have time to wait for the log backup to complete, but we rather just want to truncate and shrink the log as soon as possible. In earlier versions of SQL Server, it was possible to execute the following statement:

```
Backup log DB1 with no_log;
```

With no_log option directs SQL Server to perform just log truncation. In SQL Server 2008, this option is no longer supported, but the same effect can be achieved by changing database recovery model from full to simple. This action is potentially dangerous because it deletes log records that are not saved in any backup copy so it can become impossible to recover database to any point in time after these log records were created (Spenik & Sledge, 2003). Therefore, in such a situation, it is strongly recommended to make a full backup copy immediately after transaction log is shrunk and database becomes available again.

5. PREVENTING TRANSACTION LOG FROM GROWING TOO LARGE

Long lasting or uncommitted transactions are the main cause of rapid file growth because they do not allow database engine to truncate transaction log (Microsoft, 2008). If database is in full recovery model, which is under default settings, it is necessary to set up a backup strategy that includes making transaction log backup copies (Microsoft, 2005). This is the only way to truncate transaction log and force it to reuse its space. Backing up transaction log will not completely stop automatic transaction log growth by itself, but it is still the necessary measure that must be taken.

If we switch our database to simple recovery model, then we do not have to take care of backing up transaction logs because checkpoints truncate transaction log automatically and thus keep it from growing too much. However, we have to be aware that we lose point-in-time recoverability by working in simple recovery model. If point-in-time recoverability is not needed, it is highly recommended to switch the database to simple recovery model. Otherwise, appropriate backup strategy, which includes transaction log backups, has to be set up for the database.

6. CONCLUSION

Under default settings, SQL Server databases operate in full recovery model, which offers point-in-time recoverability but also requires proper database maintenance in order to prevent transaction log file from growing too large. The most important is to set up a backup strategy that includes taking regular transaction log backups. This is the only way transaction log can be truncated, which in turn enables it to reuse its space. If point-in-time recoverability is not necessary, it is recommended to switch the database to simple recovery model where database engine itself automatically truncates transaction log at every checkpoint. However, these steps cannot completely solve the problem with transaction log growth because things like long lasting transactions can disable log truncation. Therefore, database has to be appropriately monitored. It is advisable to set up notifications to alert database administrators when transaction log size expands over predefined value.

7. REFERENCES

Microsoft (2005). How to stop the transaction log of a SQL Server database from growing unexpectedly, *Available from:* http://support.microsoft.com/kb/873235, *Accessed:* 2010-05-18

Microsoft (2008). A transaction log grows unexpectedly or becomes full on a computer that is running SQL Server, *Available from:* http://support.microsoft.com/kb/317375/, *Accessed:* 2010-05-18

Microsoft (2009). Write-ahead Transaction Log, *SQL Server Books Online*, November 2009, *Available from:* http://msdn.microsoft.com/en-us/library/ms186259.aspx, *Accessed:* 2010-05-18

Mullins, C. (2002). *Database Administration, The Complete Guide to Practices and Procedures*, Addison-Wesley Professional, ISBN 0201741296, Canada