SQL ServerCentral.com

The World's Largest Community of SQL Server Professionals

Inside the SQL Server Transaction Log

Brad M. McGehee
Director of DBA Education
Red Gate Software
www.bradmcgehee.com/presentations

My Assumptions About You

- You are probably a DBA (production or developer) who has at least one year’s experience.
- You have a basic understanding of how the SQL Server data cache works.
- You understand the ACID properties of databases.
- You understand the differences between the full, bulk-insert, and simple recovery models.
- You understand how to perform full, differential, and transaction log backups.

What We Are Going to Learn Today

- Why Does the Transaction Log Exist
- How the Transaction Log Works
- How Are Log Records Written to the Transaction Log
- How the Transaction Log Can Become an IO Bottleneck
- How to Determine if the Transaction Log is a Bottleneck
- How to Deal with Transaction Log Bottlenecks and Boost Performance

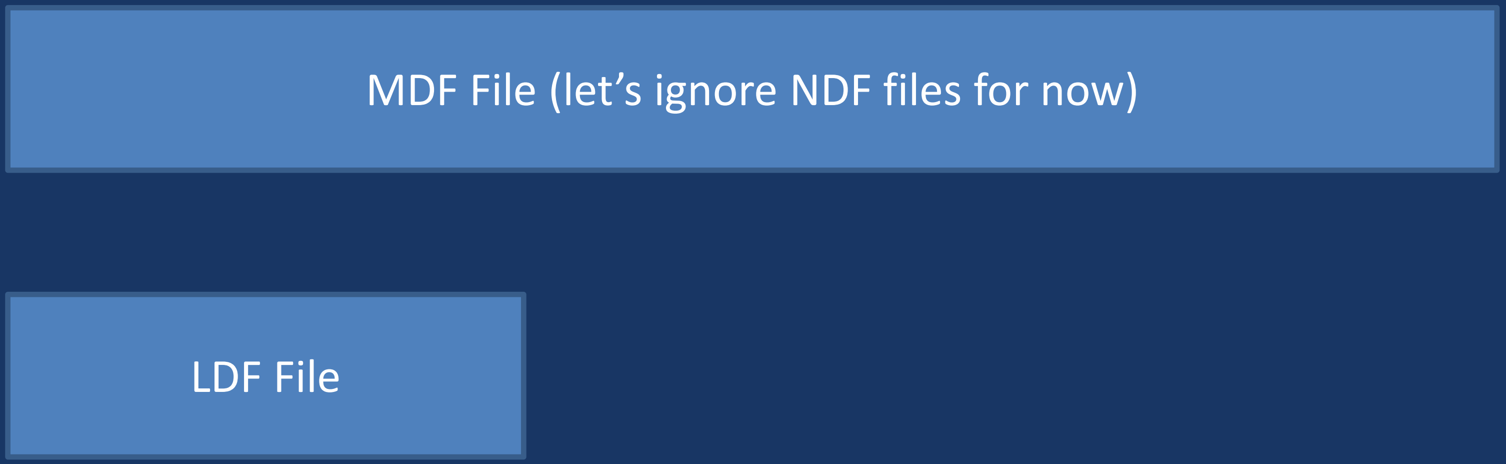
Why Does The Transaction Log Exist

- The transaction log stores a **record of all the data modifications** performed in a database.
- In the event of an unexpected shut-down, the data in the transaction log can be used during **recovery** to **roll forward any transactions completed**, but not written to the database file on disk at the time of the shut-down.
- In addition, any **uncompleted transactions** that were partially written to the database file on disk before the failure can be **rolled back** during recovery.
- Both of these actions ensure **database integrity** and the ACID properties of transactions. **This is the reason the Transaction Log exists**.

How the Transaction Log Works

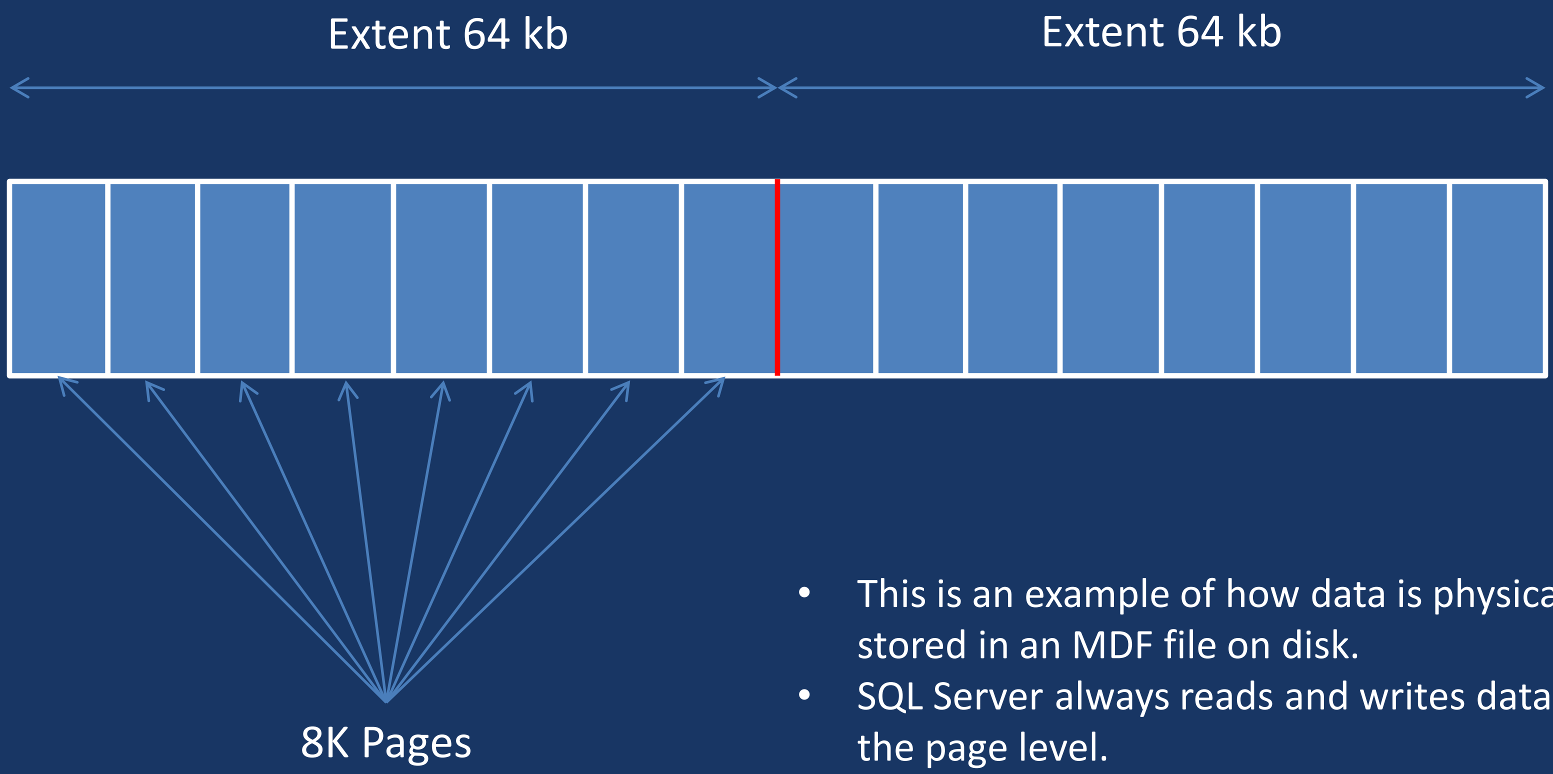
- In the following slides, I am going to start out with the **big picture** of how the transaction log works.
- For the purposes of my explanation, I am going to assume that the database uses the **full recovery model** and that **transaction log backups are performed** periodically.
- After discussing the big picture, I will provide a **specific example**, with more detail, so you better understand what happens under the covers.
- **I won't be covering every detail** of how the transaction log works, as there is not enough time. See *Inside Microsoft SQL Server 2008* for more details.

How We Often Think of Database and T-Log Files



This is a logical view, not a physical view of the MDF and LDF files.

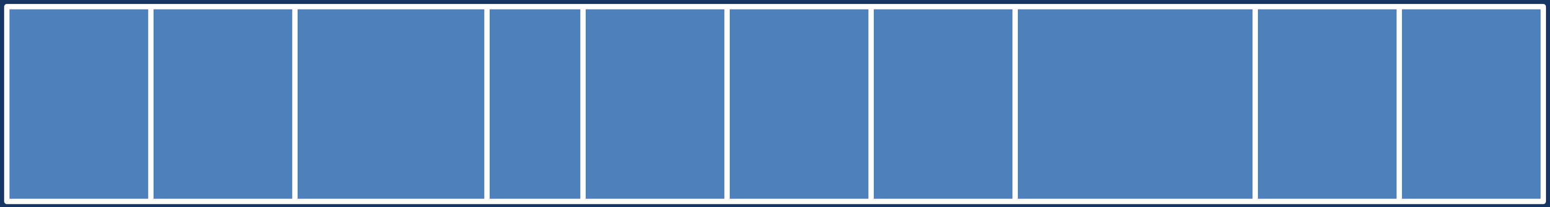
Looking at a MDF File in Detail



- This is an example of how data is physically stored in an MDF file on disk.
- SQL Server always reads and writes data at the page level.
- Some data pages may be full, others partially full, and others empty.
- Data may be written randomly or sequentially.

Looking at a LDF File in Detail

LDF files are not physically divided into pages like MDF files. Instead, they are divided into **virtual log files** (VLFs), often of uneven sizes from small to large. They are used to determine which parts of a log can be **set to reusable** after a transaction log backup. SQL Server creates the VLFs and determines their sizes. **LDF files are circular, and don't grow if they don't have to**.

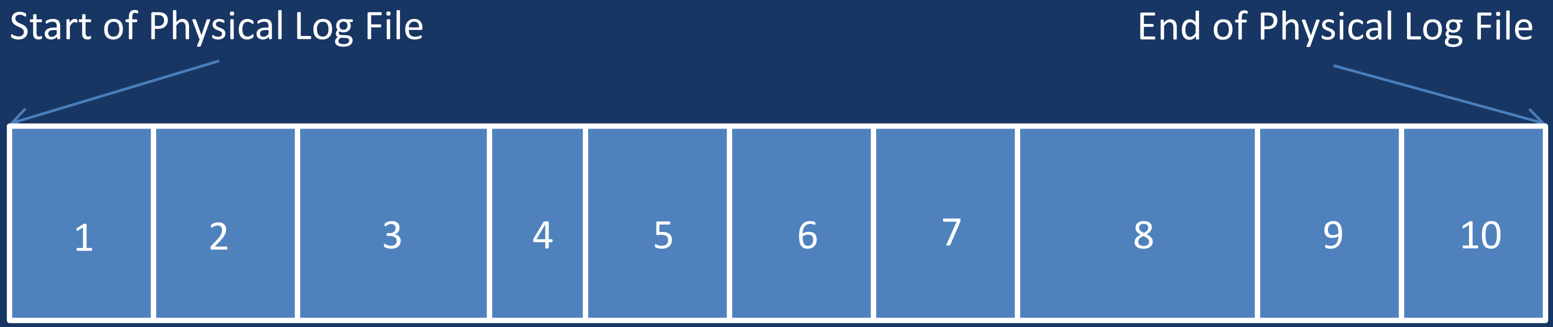


Unlike database pages, which hold rows of data, **VLFs store log records**. Log records are **written sequentially** in the log file, into available VLFs as needed, in a circular fashion. In other words, multiple log records for a single transaction can overflow from one VLF to another. (DEMO DBCC LOGINFO)

What are Log Records

- **Every modification** to the database creates multiple log records.
- Log records describe **what is being changed** and **what the change is**.
- A log record's **size will vary**, depending on the nature of the change.
- Log records are **written sequentially** (although mixed with other log records) to the log file and stored in VLFs, as previously discussed.
- Besides log records, **extra space in the log is reserved** in case a roll back occurs and **compensation log records** need to be written to the log. This is one reason why logging takes up more space that the actual modification.
- Log records are assigned **LSNs** (Log Sequence Number), an **ever-increasing, three-part number** that **uniquely defines the position of a log record within the transaction log**, and they are critical as the are used to identify which log records need to be rolled back or rolled forward during recovery.(DEMO DBCC LOG).

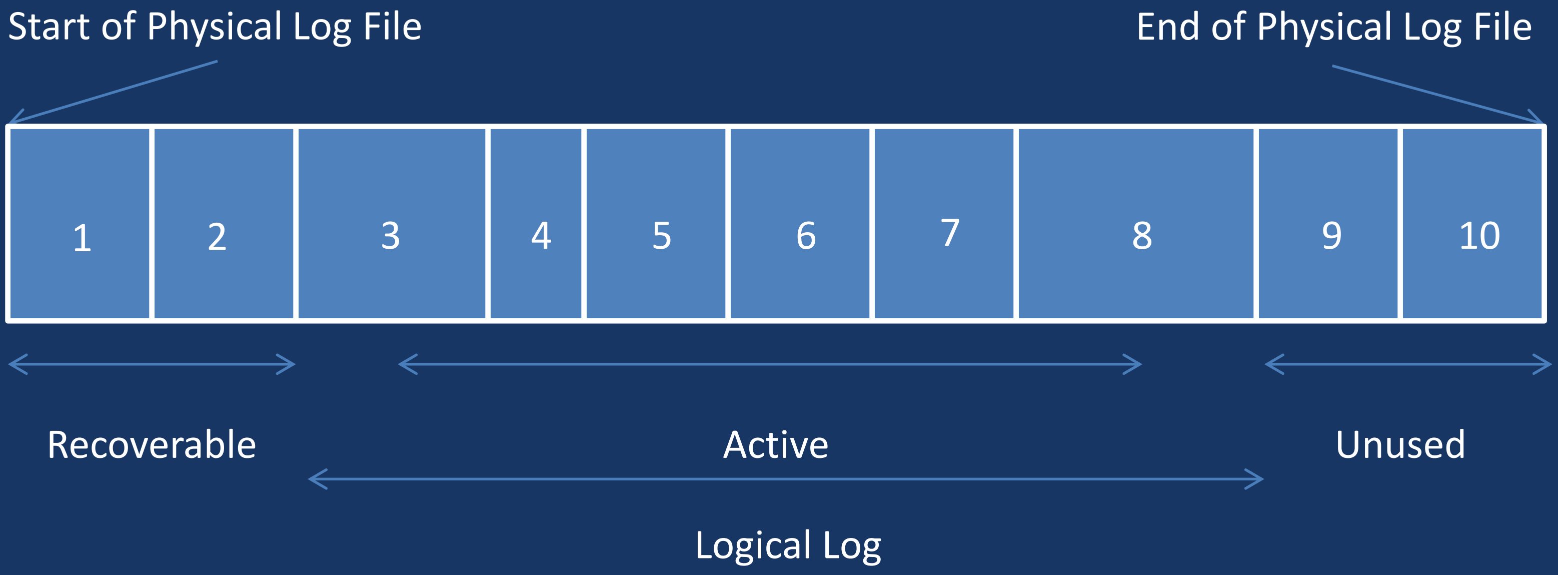
VLFs Can Be in One of Four States



A VLF can be in any one of four different states:

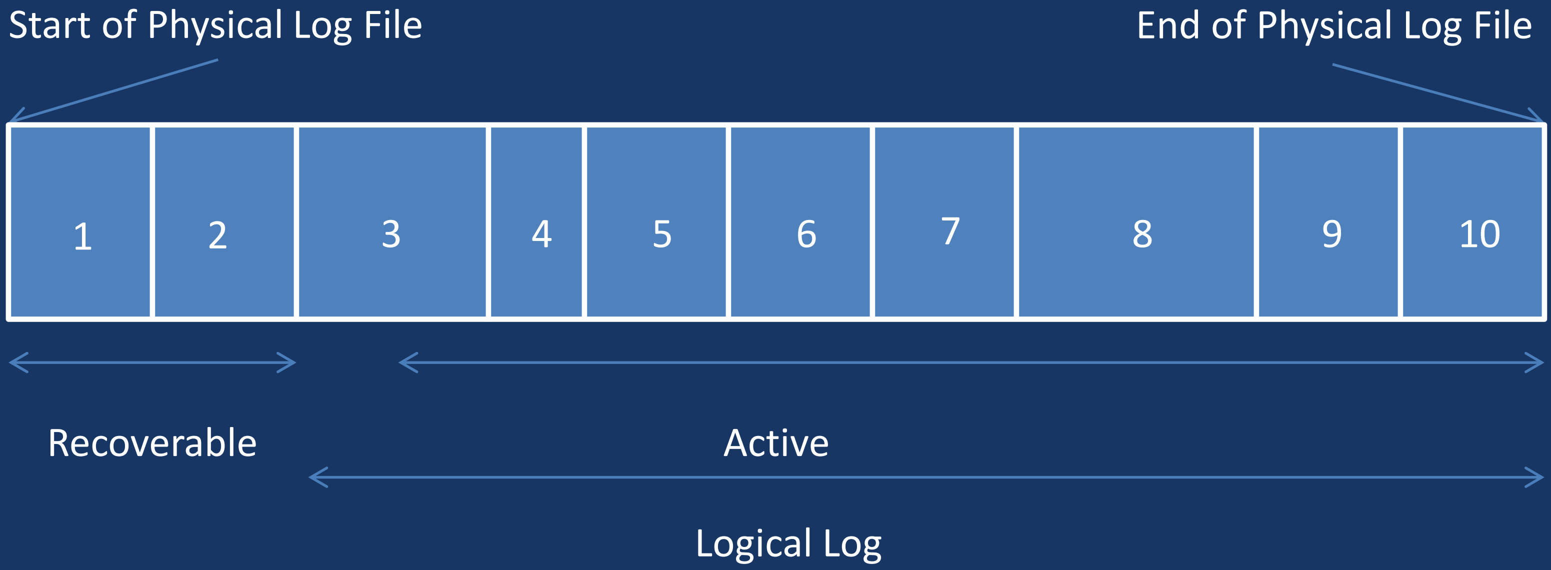
- **Active**: Includes **at least one active log record** that is part of the active log. Active records still need to be used for some purpose. (e.g. active transaction, replication, mirroring). Log backup leaves it unaffected.
- **Recoverable**: No active log records, but **not available**. Space cannot be reused until transaction log backup occurs.
- **Reusable**: Inactive log records. **Space can be reused**.
- **Unused**: **No log records** have ever been recorded in it. VLF is zeroed out.

Simple Transaction Log Example



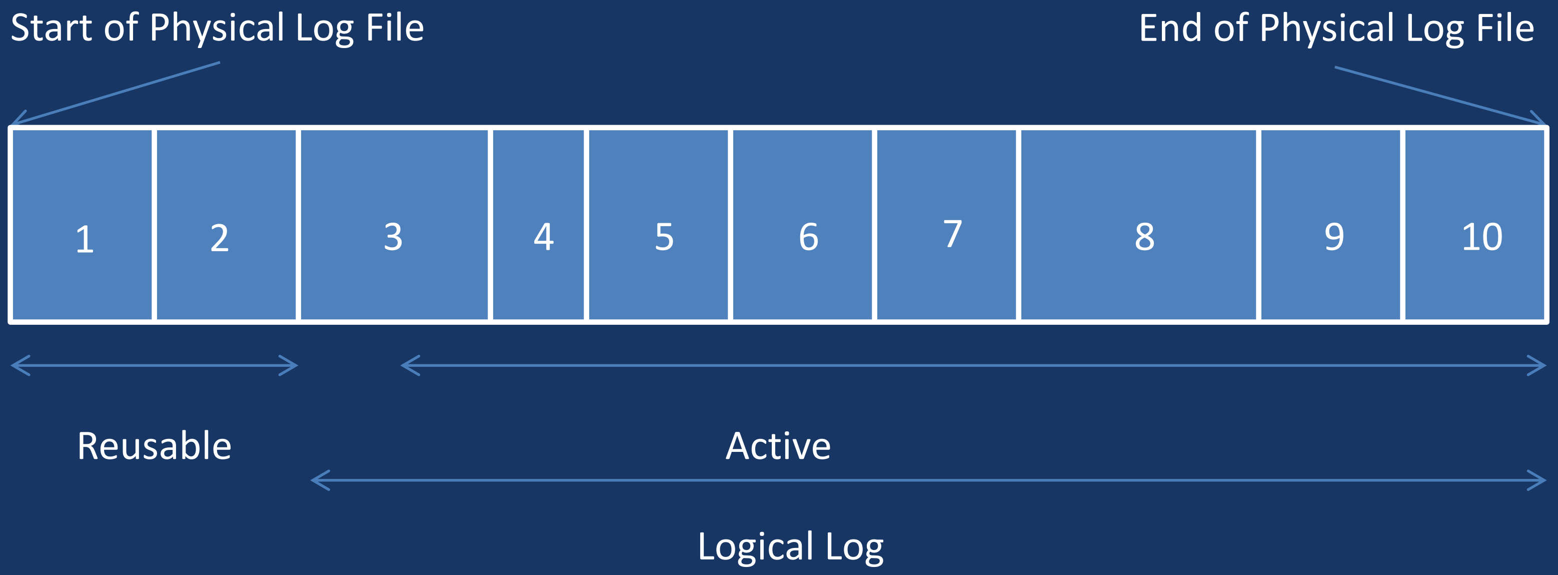
- Let’s assume the transaction log is in the above state.
- The **logical log** is the part of the log that includes the first active VLF through the last active VLF.

Log Records Are Added



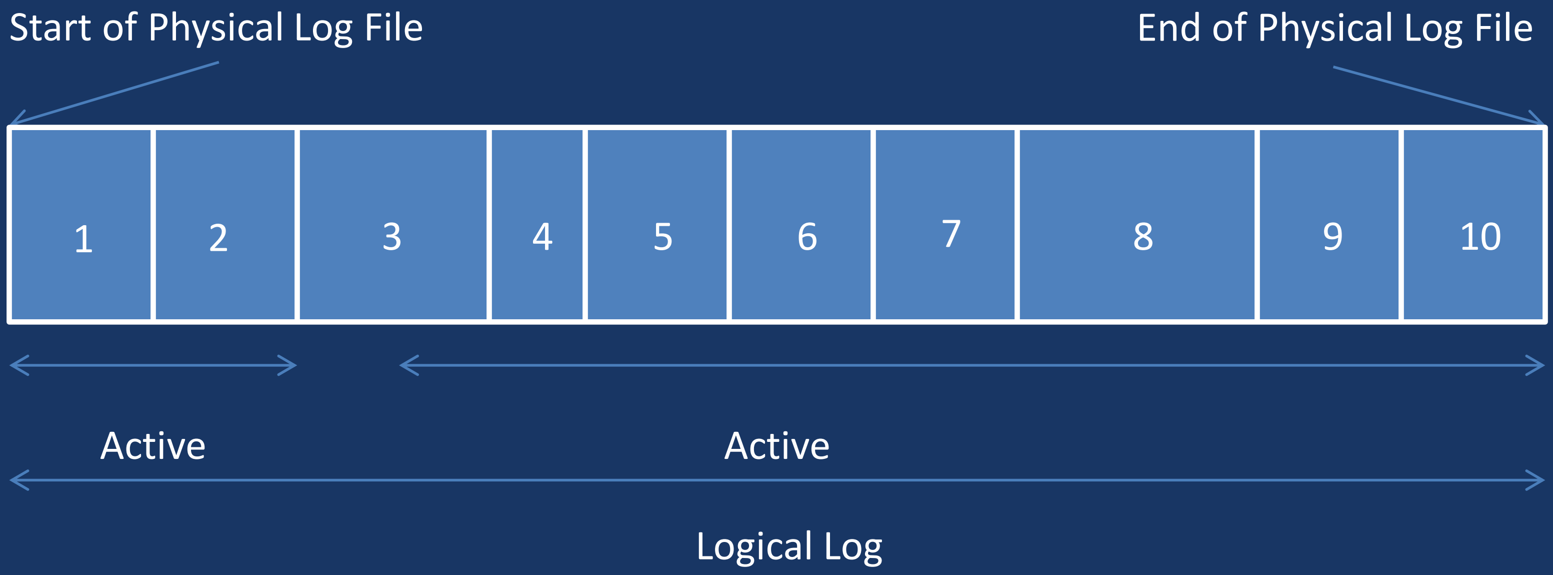
- Let’s assume that **data has been added** and the transaction log has grown to the full size of the physical LDF file.

Transaction Log Backup is Performed



- Let’s **assume that a transaction log backup** was performed.
- Notice how **recoverable VLFs (1-2) change to reusable** VLFs.
- Note that **VLF 3 is still active**, as it still has active log records.

Log Records Added After Log Backup

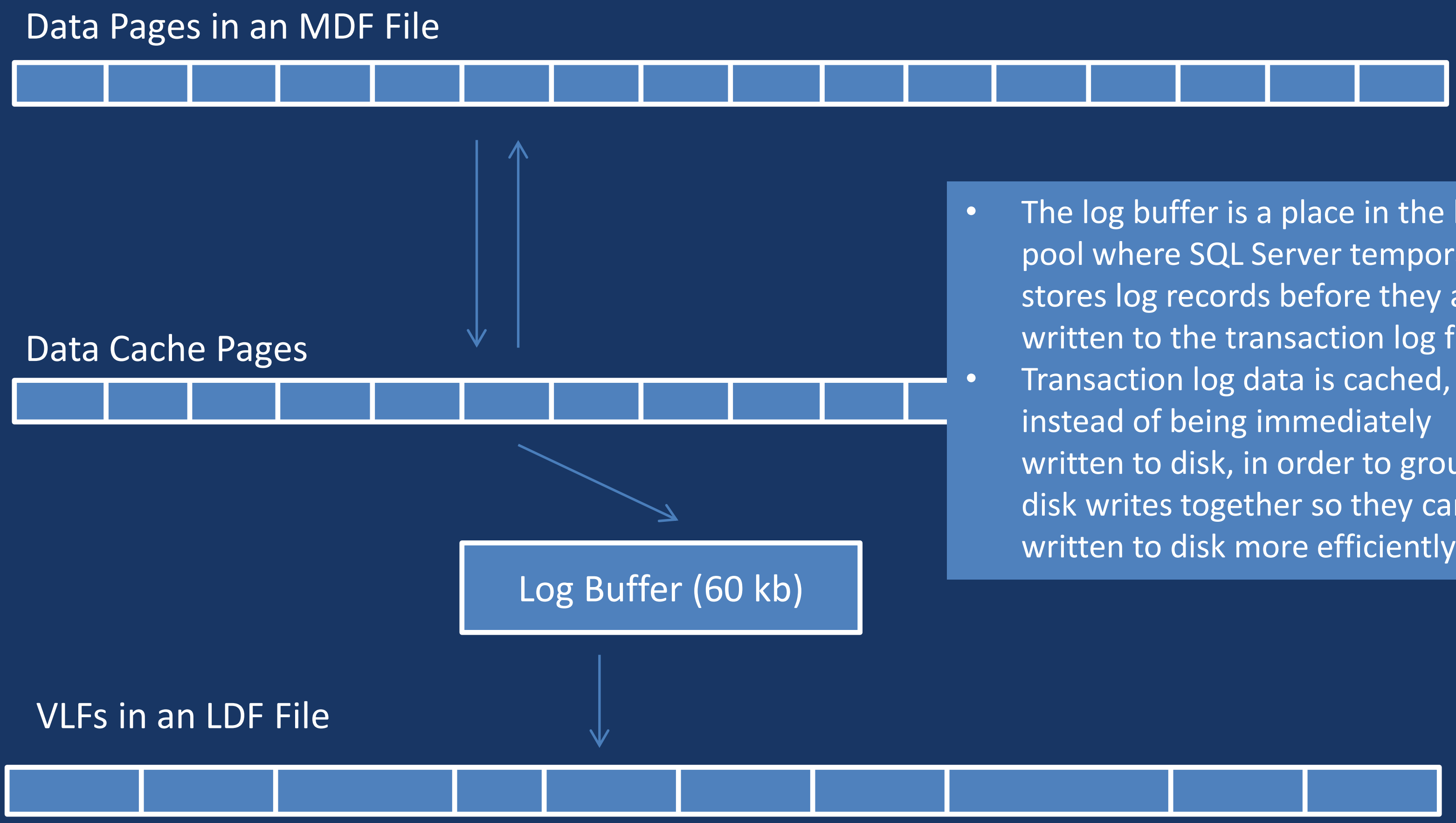


- Let’s assume that more **log records have been added**.
- If reusable VLFs exist, then the **transaction log wraps** around from end to front.
- What would happen if **more data were added** now?

How Are Log Records Written to the Transaction Log

- Now that we know a little about how the transaction log works internally, we need to **learn how transaction log records are written to the transaction log**.
- It is important to understand how log records are written to the LDF file, as how this works can directly **affect the performance** of a SQL Server instance.

Introducing the Log Buffer



- The log buffer is a place in the buffer pool where SQL Server temporarily stores log records before they are written to the transaction log file.
- Transaction log data is cached, instead of being immediately written to disk, in order to group disk writes together so they can be written to disk more efficiently.

Following the Life of a Transaction Through to the Transaction Log

- Now that we have a basic understanding of the architecture, **let's look at a specific example**.
- Let's assume that we have a **transaction that wants to INSERT a single row** into a table.
- To keep things simple, let's assume there is **no other activity** going on and that the **data cache and log buffer are empty**.

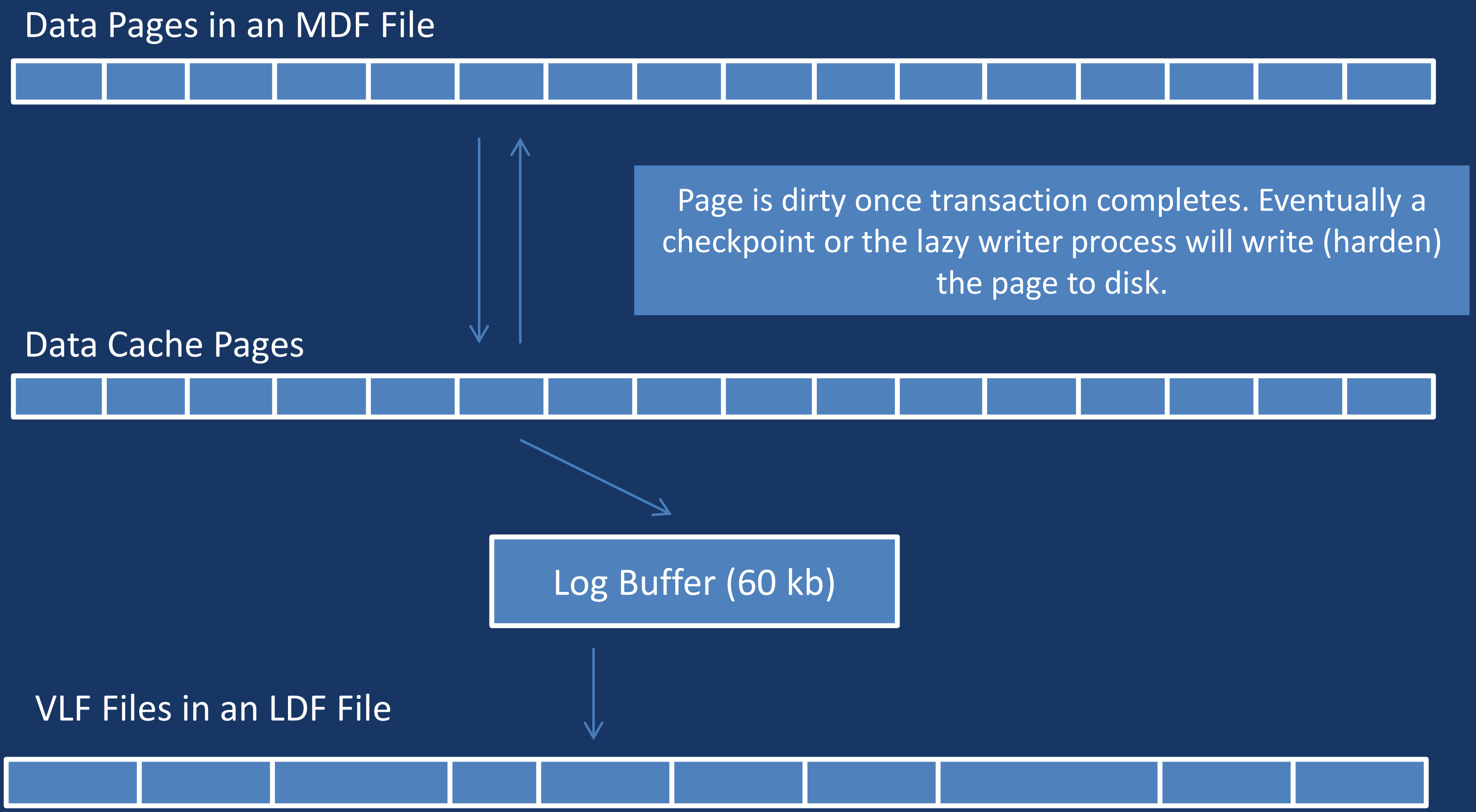
Our Example #1

- When a transaction begins, here’s what happens:
  - A **BeginTran log record** is written to the log buffer.
  - A **data page is retrieved** from disk and put into a page of the data cache.
  - SQL Server then *creates an* **INSERT log record**, which is then added to the log buffer.
  - The **new row is then INSERTed into the data page** stored in the data cache and the page is marked as being dirty.

Our Example #2

- Now that the row has been inserted, a **Commit log record** is created and written to the log buffer, and all of the log records associated with the transaction are now **flushed (written) to the transaction log file** from the log buffer.
- This process is known as **write-ahead logging**. In other words, **all data modifications must first be made to the transaction log** before it is considered a complete transaction.
- Now that the **transaction is complete,** the client is notified that the batch completed successfully.

A Visual Example



Page is dirty once transaction completes. Eventually a checkpoint or the lazy writer process will write (harden) the page to disk.

How the Transaction Log Can Become an IO Bottleneck

- As **all data modifications** have to be **written** to the transaction log, this contributes heavily to disk IO, potentially causing **performance bottlenecks**.
- What you may not know is that not only is the **transaction log written to, it is also read**.
- Let’s take a more detailed look at what data causes **writes and reads** to the transaction log, and see how it affects performance.

Common Types of Log Write Activity

- DML activity
- Index maintenance as a result of DML activity
- Page splitting as a result of DML activity
- Auto Stats creation and updating
- DDL activity
- Database maintenance, such as rebuilding or reorganizing indexes, updating statistics, etc.

Read Log Activity #1

- Every time a transaction has to be **rolled back**, log records have to be read so the roll back can occur.
- Creating a **database snapshot** requires crash recovery to be run, which reads the transaction log.
- Running **DBCC CHECKDB** creates a database snapshot as part of its internal process, and as we just noted, creating a database snapshot requires crash recovery to be run, and the transaction log read.
- Any kind of **backup**--full, differential, or log--all require the transaction log to be read.

Read Log Activity #2

- If a database is in the simple recovery mode, every time a **checkpoint** occurs, the transaction log has to be read.
- **Transactional replication** reads the transaction log in order to move changes from the publisher to the distributor.
- Using **Change Data Capture** uses the transactional replication log reader to track changes, which in turn reads the transaction log.
- **Database mirroring** reads the transaction log in order to move changes from the primary to the mirror.

How to Determine if the Transaction Log is a Bottleneck

- A variety of **tools and metrics** can be used to help diagnose if the transaction log has become an IO bottleneck.
- I will mention several, but will focus on just **two Performance Monitor counters**, as I feel these two are the best guide to determining if your transaction logs are incurring IO bottlenecks.

These Tools Can Be Used

- DBCC OPENTRAN
- DBCC SQLPERF(LOGSPACE)
- fn_virtualfilestats
- sys.dm_os_wait_stats
- sys.dm_io_virtual_file_stats
- sys.dm_io_pending_io_requests
- Performance Monitor Counters:
  - **SQL Server:Databases** (Log Bytes Flushed/sec, Log Flushes/sec, Log Flush Wait Time)
  - **Average Disk sec/Read** (The average time, in milliseconds, of a read of data from disk; read latency)
  - **Average Disk sec/Write** (The average time, in milliseconds, of a write of data to disk; write latency)

http://sqlcat.com/technicalnotes/archive/2008/12/09/diagnosing-transaction-log-performance-issues-and-limits-of-the-log-manager.aspx

When Does an IO Bottleneck Occur

- The **Average Disk sec/Read** and the **Average Disk sec/Write** counters provide you with the disk latency for the array where your transaction log(s) are located. Below is a chart I use to help me determine if IO is a problem.
  - Less than 5 ms = excellent (**this should be your goal**)
  - Between 5- 10 ms = good
  - Between 10-20 ms = okay
  - Between 20-50 ms = slow
  - Greater than 50-100 ms = potentially serious IO bottleneck
  - Greater than 100 ms = definite IO bottleneck
- Like any recommendations, the **numbers above are generic** and may not fit your particular environment.

How to Deal with Transaction Log Bottlenecks

- There are two ways to deal with transaction log bottlenecks:
  - **Prevent them in the first place**. Implementing best practices before problems arise is much easier than after the fact.
  - If they have been identified **after the fact**, implement as many transaction log best practices as you can, but your options are often more limited, and reboots required.
- Let’s discuss some best practices that you might want to consider implementing in order to prevent or reduce transaction log IO bottlenecks.

Managing Virtual Log Files: Too Many

- When the number of VLFs in a transaction log become too many, called VLF fragmentation, **performance can suffer greatly**. This includes DML, backup, and recovery performance.
- At a point of about **100-200 VLFs** or so, log performance can begin to degrade.
- To find out how many VLFs you have in a log file, run this code: "**DBCC LOGINFO** (database_name)". The number of rows returned is the number of VLFs in the database's log file.

Managing Virtual Log Files: Too Many

- If the number of VLFs **exceeds 100-200** (depending on how large the log file is), consider manually shrinking the log file, and then manually growing it to create the "optimal" number of VLFs for your environment.
- The main reason there are **too many VLFs** is because **autogrowth is used** to grow the LDF file. A big mistake.
- By default, the **number of VLFs created** is:
  - Amounts less than 64 MB = 4 VLFs
  - Amounts of 64 MB to less than 1 GB = 8 VLFs
  - Chunks of 1 GB or larger = 16 VLFs
- For more information, see:
  - http://sqlskills.com/blogs/kimberly/post/8-Steps-to-better-Transaction-Log-throughput.aspx
  - http://www.sqlskills.com/BLOGS/KIMBERLY/post/Transaction-Log-VLFs-too-many-or-too-few.aspx

Managing Virtual Log Files: Too Few

- If the log file will be 16 GB or larger, it is recommended that they are **manually grown 4 GB** at a time to ensure that the VLFs are not so large that it takes a long time for a transaction log to be cleared.
- In other words, **you can have too few VLFs**, which can prevent VLFs from being freed up on a timely basis for transaction log backups.
- In large databases, try to **keep VLFs no larger than 500 MB** or so.
  - EX: 16 GB created in one shot equals 16 VLFs of 1 GB each
  - EX: 16 GB created in two shots equals 32 VLFs of 500 MB each

Remove Physical File Fragmentation

- Use the Windows **defrag.exe** command to see if the array with your transaction logs has physical file fragmentation.
- If it does, then use the same tool, or other defragging tools, to **remove it**.
- Unfortunately, you have to **take SQL Server down** to safely defrag the files.
- The best way to deal with physical file fragmentation is to **prevent it** from happening in the first place, which is discussed next.

Preallocate Transaction Log File Size

- **Pre-sizing your log files** to their expected size offers several benefits as it prevents autogrowth from kicking in, which can cause problems. Some of the benefits include:
  - Reduces physical file fragmentation.
  - Prevents too many VLFs from being created. In other words, you control how many VLFs are created.
  - Prevents unexpected time delays (blocking) as the transaction log is growing because of autogrowth.

Disk Partition Alignment

- For optimum performance, the partitions on your disk need to be **properly aligned**, or you risk losing a lot of your IO subsystem's performance.
- If you are not familiar with this topic, read the Microsoft white paper *Disk Partition Alignment Best Practices for SQL Server* by Jimmy May and Denny Lee. It is available at: http://msdn.microsoft.com/en-us/library/dd758814(v=sql.100).aspx.
- Most commonly a problem running on **Windows 2003, or hardware that has been upgraded from Windows 2003 to Windows 2008.**

Select a Fast IO Subsystem

- Since we know the potential negative effect of transaction logs on the overall performance of SQL Server, we want to ensure that they run on the **fastest IO subsystem** possible, given your budget restrictions.
- Ideally, the transaction log should be located on a **RAID 10** array (RAID 1 if you can't afford RAID 10).
- In some very high-end OLTP systems, some companies have been using **RAID 1** SSDs (Solid State Drives) or Fusion-IO devices for maximum performance.

Separate Data and Log Files

- You have probably heard this advice a hundred times, but it is important to **separate your data (MDF, NDF) and log (LDF) files on separate arrays** in order to reduce IO contention. This is one of the easiest things you can do, and is effective in reducing IO contention between data and log files.
- In a perfect world with an unlimited budget, **each log file should be on its own array**, as mixing multiple log files on the same array can introduce a lot of random reads and writes.
- The **more log files there are** on the same shared array, the bigger this problem becomes.
- At the very least, put high-performance, mission critical logs on their **own array**.

Use Standard Performance Tuning Techniques

- Only perform as much DML or DDL as required.
- Keep transactions short.
- Minimize the possibility that a transaction has to be rolled back. Rollback sooner than later.
- Eliminate redundant or unused indexes (including indexed views), as they all need to maintained, and maintenance requires log activity.
- Take steps to minimize page splitting. For example, use a monotonically increasing clustered index key for each of your tables, and select a fill factor that minimizes internal page splitting.

Perform Transaction Log Backups Often

- Every time a **transaction log backup is made, unused portions of it are cleared out** and can be used for new logging.
- The more often you perform transaction log backups, **the quicker they take**, helping to reduce overhead on the transaction log.
- If you don’t back up the transaction logs often, and they grow large, there will be a **larger IO hit** during the transaction log backup operation.
- So, **smaller and more frequent transaction log backups are preferred** over larger and less frequent transaction log backups.

Use Minimally Logged Operations If Appropriate

- While all database modifications are logged in SQL Server, some database modifications can be **minimally logged**, which incur less IO overhead than fully logged operations.
- These include **BCP, BULK INSERT, INSERT INTO, SELECT INTO**, among several others.
- For these operations to be minimally logged, the database has to be using the **bulk-logged or simple recovery** models, among other requirements.

Transactional Replication & Mirroring

- If you use either **transactional replication** or **mirroring**, the transaction log is used to move transactions between databases.
- If transactional replication or mirroring **can’t keep up**, log records that haven’t been moved remain active, taking up space in the transaction log.
- If you use either of these, **plan for a large transaction log**, and try to ensure that replication or mirroring don’t get behind. For example, ensure that network connection between the servers is fast enough to keep up.

Schedule Database Maintenance During Slow Times

- Full and differential database backups, index rebuilds, index reorganization, statistics updates, and DBCC CHECKDB, all can **contribute to transaction log activity**.
- Consider performing these operations at times of the day **when your server is not so busy**, and be sure you **don't overlap** running them at the same time.
- This will help to **spread out transaction log IO** throughout the day, helping to minimize potential bottlenecks.

Take Aways From This Session

- Transaction logs play a **critical role** in SQL Server, and because of this, you need to pay attention to them.
- Transaction logs can become an **IO bottleneck** for busy databases because all activity is logged.
- By **following best practices**, you can help reduce or eliminate any transaction log bottlenecks, helping to boost the performance of your servers.
- Your SQL Servers may or may not be experiencing a transaction log bottleneck, but **you won't know until you check for yourself**.
- If you don't have a transaction log IO bottleneck, don't be complacent. **Be a proactive DBA** and implement as many of the above strategies to mitigate any potential future transaction log bottlenecks.

Find Out More

- Free E-books on SQL Server:
  - www.sqlservercentral.com/Books
- Check these websites out:
  - www.SQLServerCentral.com
  - www.Simple-Talk.com
- Blogs:
  - www.bradmcgehee.com
  - www.twitter.com/bradmcgehee
- Contact me at:
  - bradmcgehee@hotmail.com