DATABASE JOURNAL
The Knowledge Center for Database Professionals

Finding the Source of Your SQL Server Database I/O

October 31, 2011

When you are considering performance tuning, what do you look for? There are many things that can affect SQL Server performance: CPU usage, I/O, memory utilization, bad execution plans, inappropriate or missing indexes, and so on. The one thing I tend to find most often when a query is slow is due to the amount of I/O the query performs. When a query is slow because of I/O it could be because of bad hardware, a bad execution plan, but normally it is a poor database design or the lack of appropriate indexes to help resolve the query with minimal I/O. Where do you go looking when you want to measure or improve I/O, or find those queries that are using lots of I/O? In this article I will be exploring methods of measuring and identifying I/O related performance information for your database instances.

Which database is doing all the I/O?

When you start to research where all your I/O is going, where do you start? I like to first identify which database is using all the I/O. Having an idea of the I/O performance profile of each of my databases helps me determine where I need to focus my attention.

There are a number of different ways to identify which databases are using all the I/O. But I'm going to show you how to use a Dynamic Management Function (DMF) to identify I/O by databases. Dynamic Management Views (DMVs) and DMFs are lightweight tools that are available out of the box when you install SQL Server. They were first introduced in SQL Server 2005, and then were extended and improved with the introduction of SQL Server 2008. There are many different DMVs and DMFs available within SQL Server.

DMVs and DMFs collect statistical information from the time SQL Server starts up until the time it shuts down. The information collected by DMFs and DMVs is not persistent, meaning the information that SQL Server is collecting is lost once SQL Server stops. Some dynamic management information can also be reset by various activities that might occur while SQL Server is running. The DMF that I am going to use to show how much I/O each database has performed since SQL Server started up is sys.dm_io_virtual_file_stats.

The sys.dm_io_virtual_file_stats DMF tracks both read and write I/O information about each database file. You can use this DMF to determine which database is doing the most I/O. To demonstrate how to use the DMF let's look at the TSQL code in Listing 1.

Listing 1: Shows the total I/O for each database

```
SELECT name AS 'Database Name'
      ,SUM(num_of_reads) AS 'Number of Read'
      ,SUM(num_of_writes) AS 'Number of Writes'
FROM sys.dm_io_virtual_file_stats(NULL, NULL) I
  INNER JOIN sys.databases D
      ON I.database_id = d.database_id
GROUP BY name ORDER BY 'Number of Read' DESC;
```

When I run this command against one of my instances I get the following output.

Report 1: Output produced when running code in Listing 1

```
Database Name             Number of Read       Number of Writes
------------------------- -------------------- --------------------
```

```
JUMY                      6782123             13485
msdb                      1655678             12134
RSMS                      1151234             135466
tempdb                    743257              590987
CIMS                      72341               11234
master                    70435               12312
SNJH                      65                  37
model                     57                  1
```

Here you can see the JUMY database has performed the most read I/Os and database tempdb has performed the most write I/Os, since my SQL Server instance started up. Notice the code in Listing 1 summarizes the num_of_reads and number_of_writes columns by the name column to produce the read and write I/Os by database. I ordered the output so the database with the most read I/Os would appear first in the output from this SELECT statement. By changing the ORDER BY clause, you could easily identify the database which is performing the most write I/Os.

Since the sys.dm_io_virtual_file_stats DMF tracks the number of I/Os against each database since SQL Server started up, it may not show an accurate picture of which database is currently performing all the I/O. If you want to find out which database is now consuming the bulk of the I/O on an instance you can run the code in Listing 2.

Listing 2: Shows the amount of I/O performed by each database in the last 5 minutes

```
DECLARE @Sample TABLE (
  DBName varchar(128)
 ,NumberOfReads bigint
 ,NumberOfWrites bigint)

INSERT INTO @Sample
SELECT name AS 'DBName'
      ,SUM(num_of_reads) AS 'NumberOfRead'
      ,SUM(num_of_writes) AS 'NumberOfWrites'
FROM sys.dm_io_virtual_file_stats(NULL, NULL) I
  INNER JOIN sys.databases D
      ON I.database_id = d.database_id
GROUP BY name

WAITFOR DELAY '00:05:00.000';

SELECT FirstSample.DBName
      ,(SecondSample.NumberOfReads - FirstSample.NumberOfReads) AS 'Number of Reads'
      ,(SecondSample.NumberOfWrites - FirstSample.NumberOfWrites) AS 'Number of Writes'
FROM
(SELECT * FROM @Sample) FirstSample
INNER JOIN
(SELECT name AS 'DBName'
      ,SUM(num_of_reads) AS 'NumberOfReads'
      ,SUM(num_of_writes) AS 'NumberOfWrites'
FROM sys.dm_io_virtual_file_stats(NULL, NULL) I
  INNER JOIN sys.databases D
      ON I.database_id = d.database_id
GROUP BY name) AS SecondSample
ON FirstSample.DBName = SecondSample.DBName
ORDER BY 'Number of Reads' DESC;
```

The code in Listing 2 determines the amount of I/O performed by database for the last 5 minutes. In order to capture the amount of I/O performed for last 5 minutes interval the code in listing 2 needed to capture 2 I/O samples from the sys.dm_io_virtual_file_stats DMF. The first sample collected was placed into a table variable and then statistics returned from the second sample were used with the statistics in the table variable to calculate how much I/O each database used in the five minute interval.

Identifying IO by Physical Disk Drive

If you have your SQL Server databases spread out across many SQL Server drives you might like to know which drive is the busiest from an I/O perspective. This is easy to do by joining sys.dm_io_virtual_file_stats with sys.master_files so you can get the physical name of each database file. The code in listing 3 displays the number read and write IOs by drive letter.

Listing 3: Displaying I/O statistics by physical drive letter

```
SELECT left(f.physical_name, 1) AS DriveLetter,
        DATEADD(MS,sample_ms * -1, GETDATE()) AS [Start Date],
        SUM(v.num_of_writes) AS total_num_of_writes,
        SUM(v.num_of_bytes_written) AS total_num_of_bytes_written,
        SUM(v.num_of_reads) AS total_num_of_reads,
        SUM(v.num_of_bytes_read) AS total_num_of_bytes_read,
        SUM(v.size_on_disk_bytes) AS total_size_on_disk_bytes
FROM sys.master_files f
INNER JOIN sys.dm_io_virtual_file_stats(NULL, NULL) v
ON f.database_id=v.database_id and f.file_id=v.file_id
GROUP BY left(f.physical_name, 1),DATEADD(MS,sample_ms * -1, GETDATE());
```

By taking the first character of the file name that is contained in the physical_name column I can identify drive letter associated with the I/O counts provided by the sys.dm_io_virtual_file_stats DMF. I can then use this drive letter to sum up the I/O counts for all database files on the same drive letter. By using the code in Listing 3, I can determine which one of my database drives is the busiest for from a read, write, or total I/O perspective. Knowing which drive is the busiest can be helpful if you are trying to spread out your I/O evenly across all your different physical drives.

Disk Latency

Disk performance is important. The better your disks perform the better your database can perform I/O for all the different queries that are executed. Disk Latency is a way to measure how long it takes for an I/O to be completed. You can measure your disk latency by using the different io_stall columns available in sys.dm_io_virtual_file_stats DMF. The code in listing 4 demonstrates how to calculate the average read and write disk latency for different drive letters.

Listing 4: Calculated Disk Latency for your different database drives

```
SELECT  LEFT(physical_name, 1) AS drive,
        CAST(SUM(io_stall_read_ms) /
            (1.0 + SUM(num_of_reads)) AS NUMERIC(10,1))
                          AS 'avg_read_disk_latency_ms',
        CAST(SUM(io_stall_write_ms) /
            (1.0 + SUM(num_of_writes) ) AS NUMERIC(10,1))
                          AS 'avg_write_disk_latency_ms',
        CAST((SUM(io_stall)) /
            (1.0 + SUM(num_of_reads + num_of_writes)) AS NUMERIC(10,1))
                          AS 'avg_disk_latency_ms'
FROM    sys.dm_io_virtual_file_stats(NULL, NULL) AS divfs
        JOIN sys.master_files AS mf ON mf.database_id = divfs.database_id
                                   AND mf.file_id = divfs.file_id
GROUP BY LEFT(physical_name, 1)
ORDER BY avg_disk_latency_ms DESC;
```

When you run this code on your instance your disk latency information should be in-line with the disk latency numbers associated with your hard drive specifications. Typically your latency numbers should be below 30 ms (your mileage might vary depending on your hardware). If you track this information over time you can determine the trend for your disk latency. This will allow get a better picture of how your disk might be degrading in performance over time.

Identifying Those Queries that Are Using all Your I/O

I've written about how to find resource intensive queries before, but I thought it would be good to providing you with some code that can be used to determine which queries are performing lots of I/O. The code in Listing 5 uses the information in sys.dm_exec_query_stats to identify the top 25 most expensive queries by Read I/O.

Listing 5: Display the Top 25 Most expensive read I/O queries

```
SELECT TOP 25 cp.usecounts AS [execution_count]
      ,qs.total_worker_time AS CPU
      ,qs.total_elapsed_time AS ELAPSED_TIME
      ,qs.total_logical_reads AS LOGICAL_READS
      ,qs.total_logical_writes AS LOGICAL_WRITES
      ,qs.total_physical_reads AS PHYSICAL_READS
      ,SUBSTRING(text,
                   CASE WHEN statement_start_offset = 0
                          OR statement_start_offset IS NULL
                           THEN 1
                           ELSE statement_start_offset/2 + 1 END,
                   CASE WHEN statement_end_offset = 0
                          OR statement_end_offset = -1
                          OR statement_end_offset IS NULL
                           THEN LEN(text)
                           ELSE statement_end_offset/2 END -
                     CASE WHEN statement_start_offset = 0
                            OR statement_start_offset IS NULL
                             THEN 1
                             ELSE statement_start_offset/2  END + 1
                  )  AS [Statement]
FROM sys.dm_exec_query_stats qs
   join sys.dm_exec_cached_plans cp on qs.plan_handle = cp.plan_handle
   CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) st
ORDER BY qs.total_logical_reads DESC;
```

You could easily modify this code to identify the top 25 queries that perform write I/O. You would do that by ordering the results based on the total_logical_writes column.

Keep in mind, the query in Listing 5 is only able to find out query statistics for those queries that are currently in the plan cache. Once and execution plan is removed from the plan cache the query statistics are no longer available. Therefore this query may not report statistics for all queries that have been executed since SQL Server started up.

Where is all my I/O Going?

In this article, I provided you a number of scripts that you can use to investigate where all your I/O is going on a SQL Server instance. Each one of these chunks of code used a dynamic management view or function to obtain the I/O information. Remember that Dynamic Management views and functions get their information from a non-persistent data store. Therefore the queries I have provided will only show I/O information since the last time SQL Server started up.

See all article by Greg Larsen (http://www.databasejournal.com/article.php/1560691/Gregory-A-Larsen.htm)