DBMSs – Seminar 5

Performance Tuning in SQL Server – A Top-Down Approach

1. identify the top waits at the server level

* example:

- run `DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR)`

- run the following SELECT to obtain data about wait type LCK_M_S:

```
SELECT *
FROM sys.dm_os_wait_stats
WHERE wait_type = 'LCK_M_S'
```

=> wait_type: LCK_M_S waiting_tasks_count: 0 wait_time_ms:0

- run statements below, with T2 under READ COMMITTED:

T1	T2

`BEGIN TRAN`
`  UPDATE Movie`
`  SET Nominations = Nominations + 2`
`  WHERE MovieID = 1`
`  WAITFOR DELAY '00:00:05'`
`ROLLBACK TRAN`
	`SELECT * FROM Movie`

- i.e., T2 is waiting to obtain a shared lock.

- check the wait type again:

```
SELECT *
FROM sys.dm_os_wait_stats
WHERE wait_type = 'LCK_M_S'
```

=> wait_type: LCK_M_S waiting_tasks_count: 1 wait_time_ms: 2916

* this DMV allows us to identify the top waits at the server level.

2. correlate waits with queues (identify problematic resources)

* example - counter type: 65792; counter *Page life expectancy* (amount of time a page is expected to remain in the pool without a reference):

```
SELECT *
FROM sys.dm_os_performance_counters
WHERE object_name = 'MSSQL$SQLEXPRESS:Buffer Manager' AND counter_name = 'Page life
expectancy'
```

object_name	counter_name	cntr_value	cntr_type
MSSQL$SQLEXPRESS:Buffer Manager	Page life expectancy	13879	65792

* example - counter type: 537003264; counter *Buffer cache hit ratio* (percentage of page requests that are served from the cache):

```
SELECT *
FROM sys.dm_os_performance_counters
WHERE counter_name LIKE 'Buffer cache hit ratio%'
```

object_name	counter_name	cntr_value	cntr_type
MSSQL$SQLEXPRESS:Buffer Manager	Buffer cache hit ratio	90	537003264
MSSQL$SQLEXPRESS:Buffer Manager	Buffer cache hit ratio base	90	1073939712

=> meaningful value: 90/90 = 1 (ratio); or 1 * 100 = 100% (percentage).

* example - counter type: 272696576; counter *Page lookups/sec* (number of requests / second to find a page in the pool; time-based, cumulative):

```
SELECT *
FROM sys.dm_os_performance_counters
WHERE counter_name LIKE 'Page lookups/sec%'
```

* example - counter type: 1073874176 and 1073939712; counter *Update conflict ratio*:

```
SELECT *
FROM sys.dm_os_performance_counters
WHERE counter_name LIKE 'Update conflict ratio%'
```

object_name	counter_name	cntr_value	cntr_type
MSSQL$SQLEXPRESS:Transactions	Update conflict ratio	0	1073874176

MSSQL$SQLEXPRESS:Transactions	Update conflict ratio base	0	1073939712

- run the following scenario 2 times (i.e., produce 2 update conflicts); both T1 and T2 under FULL SNAPSHPOT:

T1	T2
	BEGIN TRAN SELECT Nominations FROM Movie WHERE MovieID = 1
BEGIN TRAN UPDATE Movie SET Nominations = Nominations + 2 WHERE MovieID = 1	
	UPDATE Movie SET Nominations = Nominations + 5 WHERE MovieID = 1 --T2 is suspended, since it's trying to acquire an X lock that conflicts with the X lock held by T1
COMMIT TRAN	=> error: Snapshot isolation transaction aborted due to update conflict...

- poll counter again (say 30 seconds have passed since the 1st time the statement was executed):

```
SELECT *
FROM sys.dm_os_performance_counters
WHERE counter_name LIKE 'Update conflict ratio%'
```

object_name	counter_name	cntr_value	cntr_type
MSSQL$SQLEXPRESS:Transactions	Update conflict ratio	2	1073874176
MSSQL$SQLEXPRESS:Transactions	Update conflict ratio base	4	1073939712

=> meaningful value: (2-0)/(4-0)/30; what do you think this counter represents?

3. drill down to database / file level – identify problematic database(s) and file(s) (data file / log file)

- run the following SELECT to obtain I/O information about database MyImdb (data file and log file):

```
SELECT *
FROM sys.dm_io_virtual_file_stats(DB_ID('MyImdb'), NULL)
```

	database_id	file_id	num_of_reads	num_of_bytes_read	io_stall_read_ms	num_of_writes	num_of_bytes_written	io_stall_write_ms	io_stall
1	23	1	92	753664	36	1	8192	0	36
2	23	2	11	1024000	5	6	28672	2	7

4. drill down to process level – identify problematic stored procedures, queries, etc.

* can define a trace to identify such processes;

- e.g., can trace event class *SQL:StmtCompleted* (occurs when a SQL statement has completed), run a workload of queries, then check the output of the trace; for each event in the output, the *Duration* column indicates the amount of time taken by the event;

- performance information should be aggregated by query pattern, for instance queries:

SELECT * FROM Movie WHERE MovieID = 1

and

SELECT * FROM Movie WHERE MovieID = 5

have the same pattern (a missing index on MovieID impacts both of them).

5. tune problematic queries – once problematic queries are identified, they can be tuned (e.g., one can create suitable indexes).