Checking High CPU Utilization in SQL Server

High CPU utilization in SQL Server can impact the performance of your database and affect overall system responsiveness. Identifying the cause of high CPU usage involves examining various aspects of SQL Server, including query performance, indexing, and system configuration. Here's a detailed guide on how to check and diagnose high CPU utilization in SQL Server:

1. Identifying High CPU Utilization

a. Using SQL Server Management Studio (SSMS) Performance Dashboard

SSMS includes a built-in performance dashboard that provides an overview of CPU usage:

1. Open SSMS.
2. Connect to your SQL Server instance.
3. Navigate to Management > Data Collection > Performance Dashboard.
4. Review the CPU Utilization section to see if the CPU usage is high.

b. Using SQL Server Dynamic Management Views (DMVs)

DMVs provide detailed insights into SQL Server performance and can help identify high CPU usage.

1. Query to Check Overall CPU Usage

This query helps you identify which queries or sessions are consuming the most CPU resources:

```
SELECT
    session_id,
    login_name,
    status,
    cpu_time,
    total_elapsed_time,
    program_name,
    host_name,
    nt_domain,
    nt_user_name
FROM
    sys.dm_exec_requests
ORDER BY
    cpu_time DESC;
```

2. Query to Check High CPU Usage by Index Scans

Indexes that are not optimized can lead to high CPU usage. This query helps identify expensive index scans:

```
SELECT
   OBJECT_NAME(s.[object_id]) AS TableName,
   i.name AS IndexName,
   s.index_id,
   s.[logical_reads],
   s.[reads],
   s.[writes],
   s.[cpu_time]
FROM
   sys.dm_db_index_usage_stats AS s
   INNER JOIN sys.indexes AS i
      ON s.[object_id] = i.[object_id] AND s.index_id = i.index_id
WHERE
   OBJECTPROPERTY(s.[object_id], 'IsUserTable') = 1
ORDER BY
   s.cpu_time DESC;
```

3. Query to Check CPU Usage by SQL Server Wait Statistics

Wait statistics can indicate resource bottlenecks, including high CPU usage:

```
SELECT
   wait_type,
   wait_time_ms / 1000.0 AS WaitTimeSec,
   wait_time_ms / (wait_time_ms + signal_wait_time_ms) * 100 AS WaitTimePercent
FROM
   sys.dm_os_wait_stats
ORDER BY
   wait_time_ms DESC;
```

2. Diagnosing High CPU Utilization

a. Analyzing Expensive Queries

1. Query Execution Plan: Review the execution plans of the queries consuming high CPU to identify inefficiencies. Use the Actual Execution Plan feature in SSMS.
2. Query Statistics: Use the following query to get query execution statistics:

```
   SELECT
    query_stats.query_hash,
    query_stats.execution_count,
    query_stats.total_worker_time / query_stats.execution_count AS AvgCPUTime,
    query_stats.total_worker_time AS TotalCPUTime,
    query_stats.total_elapsed_time / query_stats.execution_count AS AvgElapsedTime
 FROM
    sys.dm_exec_query_stats AS query_stats
 ORDER BY
    TotalCPUTime DESC;
```

b. Identifying Index Issues

1. Missing Indexes: Identify missing indexes that could help reduce CPU usage:

```
   SELECT
    migs.database_id,
    mid.[object_id],
    mid.index_handle,
    mid.equality_columns,
    mid.inequality_columns,
    mid.included_columns
 FROM
    sys.dm_db_missing_index_groups AS migs
    JOIN sys.dm_db_missing_index_details AS mid
    ON migs.database_id = mid.database_id
    AND migs.[object_id] = mid.[object_id]
 ORDER BY
    migs.index_handle DESC;
```

2. Fragmentation: Check for index fragmentation and rebuild or reorganize fragmented indexes:

```
  SELECT
   DB_NAME(ps.database_id) AS DatabaseName,
   OBJECT_NAME(ps.[object_id]) AS TableName,
   i.index_id,
   i.name AS IndexName,
   ps.avg_fragmentation_in_percent
 FROM
   sys.dm_db_index_physical_stats(NULL, NULL, NULL, NULL, NULL) AS ps
   INNER JOIN sys.indexes AS i
     ON ps.[object_id] = i.[object_id] AND ps.index_id = i.index_id
 WHERE
   ps.avg_fragmentation_in_percent > 30
 ORDER BY
   ps.avg_fragmentation_in_percent DESC;
```

c. Checking for Resource Contention

Resource contention can cause high CPU utilization:

1. SQL Server Wait Statistics: Review wait statistics to identify any contention issues:

```
 SELECT
   wait_type,
   wait_time_ms / 1000.0 AS WaitTimeSec,
   wait_time_ms / (wait_time_ms + signal_wait_time_ms) * 100 AS WaitTimePercent
 FROM
   sys.dm_os_wait_stats
 WHERE
   wait_time_ms > 0
 ORDER BY
   wait_time_ms DESC;
```

2. Blocking: Check for blocking sessions that might be causing CPU spikes:

```
  SELECT
   blocking_session_id AS BlockingSessionID,
   session_id AS BlockedSessionID,
   wait_type,
   wait_time,
   wait_time_ms / 1000.0 AS WaitTimeSec
 FROM
   sys.dm_exec_requests
 WHERE
   blocking_session_id <> 0
 ORDER BY
   wait_time DESC;
```

3. Mitigating High CPU Utilization

a. Query Optimization

- Rewrite Inefficient Queries: Optimize queries to reduce CPU usage.
- Update Statistics: Ensure that statistics are up to date to help the query optimizer make better decisions.

b. Index Optimization

- Create Missing Indexes: Add indexes that can reduce the need for expensive operations.
- Rebuild/Defragment Indexes: Rebuild or reorganize fragmented indexes to improve performance.

c. System Configuration

- Max Degree of Parallelism (MAXDOP): Adjust the MAXDOP setting to control how many processors SQL Server can use for parallel query execution.

```
EXEC sp_configure 'max degree of parallelism', <desired_value>;
RECONFIGURE;
```

- CPU Affinity: Configure CPU affinity to control which CPUs SQL Server can use.

d. Monitor and Tune

- Regular Monitoring: Use performance monitoring tools to continuously check CPU utilization.
- Performance Tuning: Periodically review and tune performance based on monitoring data.

4. Using SQL Server Performance Tools

a. SQL Server Profiler

Use SQL Server Profiler to capture and analyze events related to CPU usage:

1. Open SQL Server Profiler.
2. Create a new trace.
3. Add events related to query execution, such as SQL:BatchCompleted or RPC:Completed.
4. Analyze the trace to identify queries with high CPU usage.

b. Performance Monitor (PerfMon)

Use Windows Performance Monitor to monitor CPU usage and SQL Server performance counters:

1. Open Performance Monitor.
2. Add relevant counters, such as SQLServer:SQL Statistics and SQLServer:SQL Server Cache Manager.
3. Review the data to identify CPU bottlenecks.

Conclusion

High CPU utilization in SQL Server can be caused by various factors, including inefficient queries, missing indexes, and system configuration issues.

By using DMVs, performance tools, and SQL Server Agent jobs, you can identify and diagnose high CPU usage, optimize queries and indexes, and adjust system settings to mitigate performance issues.

Regular monitoring and performance tuning are essential to maintaining optimal CPU performance in SQL Server.