SQL DW Performance Best Practices (ADW In-A-Day Lab 02)

Contents

Overview

This module will walk you through a variety of tools and techniques for reviewing and improving the performance of your Azure Data Warehouse.

Pre-requisites

- Azure Portal Access
- Azure SQL Data Warehouse
- SQL Server Management Studio
- Execute Lab2Prep.sql script

Replicated Table Behavior

A replicated table has a full copy of the table accessible on each compute node. This will help minimize data movement during joins and aggregations. However, replicated tables gives best performance when they are small in size (<2GB) and relatively static (less frequent data inserts, deletes and updates). The following section explains this with an example.

The replicated table feature in SQL DW maintains a copy of the table in cache on each compute node and persistently stores the actual table data as a round-robin distribution. When there is a change to this table, the cache is invalidated, and the Replication Cache Manager is activated to rebuild the cached table on each compute node. The Replication Cache Manager is not activated until the first query is made against the table. The rebuild happens on the first access to the table. The query that triggered the cache rebuild will wait for the completion of the cache refresh before the query is processed. You can see this Replication Cache Manage activity and associated data movement in the execution steps.

In this section, we will identify the behavior of a query containing replicated table on its first access.

1. Open SQL Server Management Studio on your laptop and connect to your SQL DW instance.
2. Expand 'Databases' node in the left pane and select 'AdventureWorksDW' database.
3. Open a new query window and execute the following query. This will create a new replicated table from an existing table.

Logon as

```
CREATE TABLE [dimWeatherObservationSites_repl]
with (distribution = REPLICATE)
AS SELECT * FROM dimWeatherObservationSites
```

4. Execute the following query in the same query window.

```
SELECT ReadingTimestamp,
       a.SiteName,
       a.fpscode,
       b.StationLatitude,
       b.StationLongitude
 FROM  factWeatherMeasurements a
 JOIN  dimWeatherObservationSites_repl b
 ON a.fpscode = b.FIPSCountyCode
 WHERE a.fpscode = '06037' and ReadingTypeID = 65
 OPTION (LABEL = 'ReplicatedTableQuery')
```

5. Open a new query window and execute the following query to monitor the previous query execution. Note down the request_id and total_elapsed_time. How long it took to execute the query?

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE [label] = 'ReplicatedTableQuery';
```

6. Copy the value from request_id column into the query below and execute. This will show the distributed query (DSQL) plan for the query (What operation took the longest time to execute and why?)

```
SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = '<request_id>'
ORDER BY step_index;
```

7. Execute the following query to identify the replicated table cache build operation.

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE command like 'BuildReplicatedTableCache%';
```

8. Go back to the previous query window and execute the same query as in step 4 above.

9. Open a new query window and execute the following query to monitor the previous query execution. Note down the request_id and total_elapsed_time. Did it take the same time to execute as before, why or why not?

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE [label] = 'ReplicatedTableQuery';
```

10. Copy the value from request_id column into the query below and execute. This will show the distributed query (DSQL) plan for the query (Do you see the Broadcast Move operation as before? Why or Why not?)

```
SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = '<request_id>'
ORDER BY step_index;
```

11. Execute the following query to see if there is any new replicated table cache build operation. Do you see any **new** replicated table cache build operation?

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE command like 'BuildReplicatedTableCache%';
```

Performance Tuning

Identifying performance bottlenecks and eliminating them is important to get the best out of your Azure SQL DW instances. In this section, we will use DMVs (Dynamic Management Views) to identify bottlenecks and tune workload on SQL DW instance.

1. Open SQL Server Management Studio on your laptop and connect to you SQL DW instance

2. Expand 'Databases' node in the left pane and select 'AdventureWorksDW' database.

3. Open a new query window and copy/paste the following query. Click on the Display Estimated Execution Plan icon 🔲 and observe the output. What step is estimated to take the longest time?

```
SELECT ReadingTimestamp,
       a.fpscode,
       LAG(ReadingValue) OVER (PARTITION BY a.fpscode,
       b.StationId ORDER BY ReadingTimestamp ASC) AS
       PreviousValue, ReadingValue
FROM [staging].[STG_Hash_ReadingUnit] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
WHERE  a.fpscode like '1%'
OPTION (label = 'slow_query')
```

4. Click on 'Execute' to run the query (you may not see any results).

5. Open a new query window and execute the following query to monitor the previous query execution (How long it took to execute the query and what is the resource class that this query has been executed with?)

```
SELECT  *
FROM     sys.dm_pdw_exec_requests
WHERE [label] = 'slow_query';
```

6. Copy the value from request_id column into the query below and execute. This will show the distributed query (DSQL) plan for the query (What operation took the longest time to execute and why?)

```
SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = '<request_id>'
ORDER BY step_index;
```

7. Copy the request_id and step_index with highest elapsed time into the following query and execute. This will show the execution information of the query step.

```
SELECT * FROM sys.dm_pdw_sql_requests
WHERE request_id = '<request_id>' AND step_index = <index>;
```

8. Copy the request_id and step_index into the following query and execute. This will show information about the data movement step. What can we do to eliminate the data movement step?

```
SELECT * FROM sys. dm_pdw_dms_workers
WHERE request_id = '<request_id>' AND step_index = <index>;
```

9. Open the following script into a new query window in SSMS and click on Display Estimated Execution Plan. Do you see any data movement step in the plan? Why or Why not?

```
SELECT ReadingTimestamp,
       a.fpscode,
       LAG(ReadingValue) OVER (PARTITION BY a.fpscode,
       b.StationId ORDER BY ReadingTimestamp ASC) AS
       PreviousValue, ReadingValue
FROM [dbo].[factWeatherMeasurements] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
WHERE  a.fpscode like '1%'
OPTION (label = 'fast_query')
```

10. Click on 'Execute' to run the query (you may not see any results).

11. Repeat steps 5 – 8 to monitor this query (use 'fast_query' as label). What differences do you notice between these query executions?

Resource Class Usage

Running workload with a higher resource class can reduce concurrency and impact overall performance due to queuing.

In this section, we will learn how to identify queuing related to queries running with large resource classes and how to troubleshoot these issues.

1. Open SQL Server Management Studio on your laptop and connect to your SQL DW instance as 'usgsLoader' user.

2. Expand 'Databases' node in the left pane and select 'AdventureWorksDW' database. Open the following SQL script in new query window and Click on 'Execute' to run the query.

```
SELECT ReadingTimestamp,
       a.fpscode,
       LAG(ReadingValue) OVER (PARTITION BY a.fpscode, b.StationId
ORDER BY ReadingTimestamp ASC) AS PreviousValue, ReadingValue
FROM [dbo].[factWeatherMeasurements] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
OPTION (label = 'long_query')
```

3. While the above query running, open a new query window as 'sqladmin' user and execute the following SQL script in it.

```
SELECT ReadingTimestamp,
       a.fpscode,
       LAG(ReadingValue) OVER (PARTITION BY a.fpscode, b.StationId
ORDER BY ReadingTimestamp ASC) AS PreviousValue, ReadingValue
FROM [dbo].[factWeatherMeasurements] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
WHERE  a.fpscode like '1%'

OPTION (label = 'fast_query')
```

4. Open a new query window and execute the following query to monitor the previous query execution (Do you see any queries in 'Suspended' state, why? Hint: Notice the 'resource_class' column)

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE [status] in ('Running', 'Suspended')
```

5. Execute the following query to see what queries are waiting. (Do you see any sessions waiting, why? Hint: Notice the 'state' column with 'Queued' state and check the corresponding value in 'type' column)

```
SELECT waits.session_id,
       waits.request_id,
       requests.command,
       requests.status,
       requests.start_time,
       waits.type,
       waits.state,
       waits.object_type,
       waits.object_name
FROM   sys.dm_pdw_waits waits
    JOIN  sys.dm_pdw_exec_requests requests
    ON waits.request_id=requests.request_id
```

6. Go to the window that is running long_query script (in step 2 above) and cancel the execution (Click on ▪ icon).

7. Right-click anywhere in the query window, click on 'Connection' and select 'Change Connection'.

8. Enter 'sqladmin' for login and 'Password!1234' for password and click 'Connect'. Click on 'Execute' to re-execute the query.

9. While the above query running, go back to the query window that was running fast_query (in step 3 above) and re-execute it.

10. Open a new query window and execute the following query to monitor the previous query execution (Do you see any queries now in 'Suspended' state, why not? (Hint: Notice the 'resource_class' column)

```
SELECT *
FROM    sys.dm_pdw_exec_requests
WHERE [status] in ('Running', 'Suspended')
```

Conclusion: The long running query that was running under 'StaticRC60' resource class uses a large number of concurrency slots. This will reduce the concurrency as there are not enough available concurrency slots. So, you can not run another query even if you are running under the SmallRC. But once you change the long query to run under SmallRC, which does not reserve many available concurrency slots, you will be able to run other queries.

Result Set Caching

Result set caching has been recently introduced in Azure SQLDW. This feature will allow you to cache the final results of a query. This will help improve performance when the same query is executed multiple times, so the results can be retrieved from the cache without actually executing the query.

In this section, we will learn how to enable this feature and observe the performance of a long running query under the influence of this feature.

1. Open SQL Server Management Studio on your laptop and connect to your SQL DW instance as 'sqladmin' user.
2. Expand 'System Databases' node in the left pane and select 'master' database. Run the following statement to enable result set caching feature on your database.

```
ALTER DATABASE sqldw72926
SET RESULT_SET_CACHING ON;
```

3. Run the following query to confirm that the Result Set Caching has been enabled on your database.

```
SELECT name, is_result_set_caching_on
FROM sys.databases;
```

4. Expand 'Databases' node in the left pane, right-click on 'AdventureWorksDW' database and click on 'New Query'.

5. Execute the following query in the new query window.

```
SELECT TOP 5 ReadingTimestamp,
        a.fpscode,
        LAG(ReadingValue) OVER (PARTITION BY a.fpscode, b.StationId
ORDER BY ReadingTimestamp ASC) AS PreviousValue, ReadingValue
FROM [dbo].[factWeatherMeasurements] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
where a.fpscode > 50000
OPTION (label = 'longer_query')
```

6. Wait until the query execution completes and note down the duration.

7. Execute the same query again by clicking on the 'Execute' button. Wait until the query execution completes and note down the duration.

8. Did the second execution of the query took under a second? Why?

9. Execute the following query from the same query window. It should show both executions of the above query.

```
SELECT *
FROM sys.dm_pdw_exec_requests
WHERE [label] = 'longer_query'
```

What differences do you see between both executions of the query? (HINT: Observe 'total_elapsed_time' and 'resource_class' columns)

10. Execute the following query from the same query window. It will show the execution steps of the above query.

```
SELECT *
FROM sys.dm_pdw_request_steps
WHERE request_id in (SELECT request_id
FROM sys.dm_pdw_exec_requests
WHERE [label] = 'longer_query')
```

What differences do you see between the execution steps of both executions? (Hint: Observe the 'operation_type' and 'command' columns)

11. Slightly modify the long query by changing the value in the WHERE condition as below and re-execute the query.

```
SELECT TOP 5 ReadingTimestamp,
       a.fpscode,
       LAG(ReadingValue) OVER (PARTITION BY a.fpscode, b.StationId
ORDER BY ReadingTimestamp ASC) AS PreviousValue, ReadingValue
FROM [dbo].[factWeatherMeasurements] a
JOIN [dbo].[dimWeatherObservationSites] b
ON a.fpscode = b.fipsCountycode
where a.fpscode > 50001
OPTION (label = 'longer_query2')
```

Did it still finish executing under a second? Why or why not?

Conclusion:
The result set caching feature helps improving the performance of repeatedly executing queries by caching the final result set and retrieve this result set from the cache for the subsequent executions of the queries.