Performance Best Practices

in Mainframe Migration to .NET + SQL Server

This topic discusses some of the options for performing database queries when migrating from a Mainframe to .NET and SQL Server. Because of the architecture differences between SQL Server and Mainframe, what works well in the Mainframe world might not be a good practice when directly translated into the .NET + Microsoft SQL Server world.

In Mainframe batch processes, it is typical that workloads use one central loop over a database cursor to get the objects to be processed and then generate tens of separated queries for each object. In Mainframe architectures the application code usually executes in an environment that is very close to the database and therefore the latency of database interactions is extremely low. Because of this Mainframe applications often make liberal use of the databases and their interactions with databases often end up being very “chatty”.

However, in .NET + SQL Server world, because of the client-server architecture, in addition to query execution there are costs in network latencies, communication, serialization, and deserialization. The best practice is to reduce database roundtrips by doing batch processing.

This document describes some general performance recommendations, several query batching strategies with their pros and cons, as well as a list of helpful performance turning resources.

General Performance Recommendations

- **Cache results when possible.** Avoid querying the SQL Server more than once for the same data within the same logical transaction by caching results
    - There is an entire range of caching techniques, for example, storing data as reusable variables, or using faster in-memory key-value stores like Redis cache. These will not be discussed in this document.
- **Design efficient indexes.** Efficient indexes are critical to application and database performance. Guidance includes
    - Keep length of key short for clustered indexes. In addition, clustered indexes benefit from being created on unique or non-null columns.

    - Hash indexes are good for point-lookup (exact single-result match) but not for range scans. If the predicate in the `WHERE` clause does not specify an exact value for each column in the hash index key, the query will do a scan instead which is less efficient.

  For more information, see SQL Server Index Design Guide.

- **Use faster CPU**. It is not recommended to run applications on the same machine as SQL Server. In the extreme case when very low latency is needed, and batching is not allowed this becomes an option because it is possible to avoid the network latency by using the Shared Memory or the Named Pipe protocol. In this option, the performance bottleneck is CPU since instead of being blocked by the network transportation since most of the cost is now on inter-process communication through memory. Using more powerful CPUs running at higher clock speeds will yield better performance.
- **Excessive calls to SQL Server Dynamic Management Views can have overhead.** Monitoring performance using SQL Server Dynamic Management Views or `SqlConnection` .NET objects can have negative impacts on performance, especially when optimizing at microsecond level so they should be avoided. For lightweight performance monitoring Extended Events would be ideal.
- **Make efficient calls to Store Procedure via ADO.NET**. Some examples include
    - When executing the same stored procedure with different query arguments, create query parameter objects once and modify values of the input parameters as needed
    - Use output parameters and `SqlCommand.ExecuteNonQuery()` for single row result
    - Pass parameters with matching types to avoid costs on implicit conversions in T-SQL
    - Use appropriate T-SQL data types for parameter to avoid collation or reduce I/O. For example, data types with smaller size can reduce time spent on I/O.
- **Ensure High Performance power plan setting.** The Power management plan of Windows influences overall system performance, including SQL Server and .NET applications.
- **Use In-Memory OLTP when appropriate.** SQL Server In-Memory OLTP technology (including memory-optimized tables and natively compiled store procedures) can improve the performance of workloads on the SQL Server side. Here are some helpful links
    - In-Memory OLTP Overview
    - Memory-Optimized Tables
    - Best Practices for Calling Natively Compiled Stored Procedures

Direct Translation

During migration, changing application logic or code comes with a cost. Decisions must be made to balance between the cost of code changes and the performance gain of modifying code to adopt best practices and patterns of .NET and SQL Server. Sometimes it might cost less to directly translate Mainframe applications into .NET + SQL Server world. In this case It could be helpful to utilize automated solutions that translate and host Mainframe applications in .NET + SQL Server environments.

Approach

Direct translation of Mainframe application into .NET + SQL Server world. There are some third-party solutions available in the market that can help automate the translation and host for example COBOL + DB2 applications without needing to change or rewrite any code manually.

The database queries are done by invoking stored procedures containing database queries translated into T-SQL, using `SqlCommand.ExecuteReader()` and `DataReader` APIs from ADO.NET.

Advantage

- Translated .NET code just works without having to change or rewrite application logic or add new logic.

Things to keep in mind

- Database roundtrips introduce latency and communication overhead, even when the SQL Server is running on the same machine as the client using Shared Memory protocol.

Query Batching Strategies

When the performance gain of code changes outweighs the cost, there are some batching strategies that can improve performance.

When there is a need to execute same T-SQL query or stored procedure multiple times, each time individually with a different argument, it is better to send these queries together to server and retrieve result in a batch to reduce database roundtrips, instead of one roundtrip for each execution. There are several approaches to batch queries. These all will likely require logic/code changes to the original Mainframe applications.

Strategy 1: Do the Work in .NET Client

When the table to query against is not being updated for an extended period and its size is not huge, the .NET client application can keep a local cache of the table and query against the cache instead.

Approach

Doing one database query to retrieve all the data of a database table into .NET client then query against the local cache.

The following ADO.NET example shows usage of a `DataSet` and a `DataView` on the table with an index on the querying column. Queries are done by calling `DataView.FindRows()`.

Filling dataset and creating data view:

```
    var connection = new SqlConnection(connectionString);
    connection.Open();
    var queryString = "SELECT * from tablename";
    var adapter = new SqlDataAdapter(queryString, connection);
    adapter.Fill(dataSet, "tablename");

    dataView = new DataView(dataSet.Tables["tablename"], string.Empty, "id ASC",
DataViewRowState.OriginalRows);
```

Querying using data view:

```
    var rows = dataView.FindRows(queryParameter);
```

Advantage

- Queries run very fast against the local cache therefore unnecessary roundtrips to SQL Server are avoided.

Things to keep in mind

- The memory footprint of the program increases.
- It might not be feasible to load huge tables into memory.
- It requires additional logic to handle cache expiration policy.
- There are several ways for .NET Code to interact with SQL Server: ADO.NET, Dapper, or Entity Framework for some examples. They all have different performance characteristics and require different amounts of code changes.

Strategy 2: Do the Work in SQL server

There are cases where caching database tables doesn't make sense, for example, they are being updated frequently; or the table sizes are too big to fit in client machine's memory; or the original mainframe code is mainly doing DB2 server-side batching. In these cases, it might be better to migrate the workloads into SQL Server T-SQL and do most of the work in SQL Server.

Approach

The query to get an object to process, as well as the following queries for the object are all done in T-SQL stored procedure(s) on the SQL Server side.

Advantage

- This removes the "chatty" point-lookup queries and the communication cost associated with them.

Things to keep in mind

- Cost of code changes might be high as it is basically re-writing the business logic of COBOL applications in T-SQL.
- Although T-SQL has very good performance at accessing data. It is not the best language for general-purpose programming. T-SQL might not suit well when there is business logic performed on the retrieved object before the subsequent queries.

Strategy 3: Hybrid Client/Server Batching

Caching the database tables could lead to a large amount of memory consumption on the client side. When memory usage becomes a concern or blocker, a hybrid client/server batching can be used to reduce the memory footprint of client application, while still achieving very low cost per query.

Approach

This approach groups keys in a batch then sends them in a query to SQL Server, either using table-valued parameter (TVP) to a stored procedure, or executing `SELECT` statements where query ids are batched in the `IN` clauses, for example

```
SELECT * FROM table WHERE Id in (71, 52, 33, 412, …, 25930)
```

The table-valued parameter approach generally has better performance but `SELECT … IN (…)` statement is easier to code.

Advantage

- Less roundtrips to SQL Server than singleton queries while having dialable memory footprints smaller than that of caching everything in the client.

Things to keep in mind

- Long SQL query expressions to SQL Server could potentially increase the query execution and transportation time when using `SELECT … IN (…)` statements.
- TVP method requires setting up the parameter type in SQL Server and preparing the parameter, which adds a bit more complication to the system.

Additional Performance Tuning resources

The .NET Framework Performance document contains introduction to .NET performance design and analysis, with links to various tools and techniques.

Improving .NET Application Performance and Scalability is still a valuable resource even though it appears as retired content. The principles, rules, and guidelines of measuring, testing, and turning performance largely remain the same.

Performance Center for SQL Server provides an abundant amount of information on improving SQL Server performance.

Measuring Performance and collecting metrics help determine whether an application meets its performance goals and identifies bottlenecks. Microsoft provides general purpose profilers and analysis tools like Windows Performance Toolkit and PerfView.

Microsoft also provides PSSDIAG and SQLDiag utilities to help monitor and troubleshoot problems with applications or servers.

Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data Migration Jumpstart Team (askdmjfordmtools@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.