Parameter sniffing

Last updated by | Joao Antunes | Oct 18, 2022 at 7:40 AM PDT

Contents

Performance issue caused by parameter sniffing

What is Parameter Sniffing?

When a stored procedures is compiled or recompiled, the parameter values passed for that invocation are "sniffed" and used for cardinality estimation. The net effect is that the plan is optimized as if those specific parameter values were used as literals in the query. Take the following stored procedure as an example:

~~~
```Sql
create procedure dbo.SearchProducts
@Keyword varchar(100)
as
select * from Products where Keyword like @Keyword
```
~~~

Assume the table is has approximately 100,000 rows, and has a single-column nonclustered index on the Keyword column. Let's say you call this the first time and pass in a parameter @Keyword='Barbie%'. Suppose the number of rows in the table with a keyword starting with Barbie is very small -- perhaps just a few dozen rows. The optimizer might choose to use a query plan that uses the index on the keyword column to evaluate the LIKE, then a bookmark lookup to retrieve the other columns for the row. This index seek + bookmark lookup plan will be cached and reused for subsequent executions of the procedure.

Performance Problems Caused by Parameter Sniffing

However, at some point in the future the server must compile/recompile a new plan for the stored procedure (the prior plan may have been aged out of cache, or auto update statistics kicked in on the Products table, etc). Unfortunately, the particular execution of the proc that compiled the new plan had a @Keyword parameter of 'A%'. Suppose that the filter 'A%' returns 10% of the rows in the table. When compiling the procedure with this parameter, SQL might select a query plan that uses a full table scan. That plan would be ideal for the parameter 'A%', but would be a terrible plan for other, more selective, search criteria. Unfortunately, following the recompile, the table scan plan would also get cached and reused. The performance of subsequent executions with more typical parameter values would suffer.

Parameter sniffing allows SQL to compile a plan that is tailored to the type of parameter that is actually passed into the proc. Generally speaking, this feature allows more efficient stored proc execution plans, but a key requirement for everything to work as expected is that the parameter values used for compilation be "typical". As illustrated in this hypothetical example, when a procedure or parameterized query may occasionally be executed with an atypical parameter (data skew is often at play in these cases) you can end up caching and reusing a query plan that was optimized for an atypical parameter value.

Parameter sniffing performance problems can affect all sorts of queries, but queries that use LIKE and range conditions (like the example described above) are especially prone to this class of problem. Perf problems caused by parameter sniffing are generally considered to be By Design. One common variation on parameter sniffing-related performance problems can arise when a stored procedure changes parameter values before using them.

Fixing Parameter Sniffing Perf Problems

To workaround this problem, consider the following options:

- Use OPTION (RECOMPILE) at statement level starting SQL Server 2005. This one is prefered over WITH RECOMPILE at procedure level. If the procedure has many statements, saving can be significant as statement level recompile only impact that particular statement.
- Use WITH RECOMPILE when you create the procedure so that each invocation will compile a new plan. Yes, this will prevent plan reuse, but sometimes that is exactly what you need (in some cases you need to optimize for two or more classes of parameters, and there is no single plan that will be acceptable for all common parameter values). Check the compile time and note the rate at which the proc is executed; if compile time is small relative to the execution time and if the proc will not be executed hundreds of times per second, WITH RECOMPILE is a reasonable solution to this problem.
- On SQL 2005 you can use an OPTION (OPTIMIZE FOR...) hint. This overrides the actual parameter values (for the purposes of query compilation only), in order to provide the query optimizer with "typical" param values it should always optimize for.
- Assign the incoming parameter values to local variables and use the local variables in the query. This forces cardinality estimation to use average column density instead of the histogram. But the issue can happen again in future so it’s a workaround.
- Use DISABLE_PARAMETER_SNIFFING, it has the same effect of local variable
- Use a query hint in the stored proc, or a plan guide for parameterized queries. (USE PLAN, OPTIMIZE FOR, INDEX=...)
- If you know what parameters cause a different plan (can find the values from XML plan), build nested stored procedures for those parameter values and call them within the main proc that had the problem in the first place. It will look something like this:

```
create proc MainProc @param1 int
as
if (@param1 = 200)
exec proc200 @param1
else if (@param1 > 200)
exec ProcGt200 @param1
else
exec ProcGenera @param1
```

- Though less desirable, you can dynamically concat the SQL statement and use exec to dynamically execute the query.

Additional Info

- Here is a blog entry describing parameter sniffing: https://blog.sqlauthority.com/2019/12/19/sql-server-parameter-sniffing-simplest-example/. The blog post includes some hands-on examples that allow you to see parameter sniffing in action and explore some of these possible solutions.

- Understand [statistics] (https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver15#UpdateStatistics) and how it would impact the performance

How good have you found this content?